О.Л. Голицына, Н.В. Максимов,
И.И. Попов

БАЗЫ ДАННЫХ

3-е издание, переработанное и дополненное

Рекомендовано Учебно-методическим объединением вузов Российской Федерации по образованию в области прикладной информатики в качестве учебного пособия для студентов высших учебных заведений, обучающихся по направлению 230700 «Прикладная информатика»

ФОРУМ
МОСКВА
2012

004.65(075.8)

УДК 004.6(075.8)
ББК 32.973я73
Г60

Рецензенты:
доктор технических наук, профессор РГГУ *С.А. Косяченко*;
кандидат технических наук, доцент РГГУ *Л.Н. Сысоева*

Голицына О.Л., Максимов Н.В., Попов И.И.

Г60 Базы данных : учебное пособие / О.Л. Голицына, Н.В. Максимов, И.И. Попов. — 3-е изд., перераб. и доп. — М. : ФОРУМ, 2012. — 400 с. : ил. — (Высшее образование: бакалавриат).

ISBN 978-5-91134-630-0

В пособии рассмотрены основные подходы и направления развития систем баз данных. Анализируются классические машинно-ориентированные формы представления информации и данных. Рассматриваются типовые модели физической и логической организации данных, архитектура средств доступа к данным. Достаточно подробно представлены возможности SQL как базового языка для работы с реляционными базами данных. Большое внимание уделено проблемам моделирования и проектирования баз данных.

Предназначено для студентов вузов, обучающихся по направлению «Прикладная информатика» (ФГОС 3-го поколения), и учащихся техникумов и колледжей.

УДК 004.6(075.8)
ББК 32.973я73

ISBN 978-5-91134-630-0

Предисловие

Сегодня трудно себе представить сколько-нибудь значимую информационную систему, которая бы не имела в качестве основы (или важной составляющей) базу данных. Концепции и технологии баз данных складывались постепенно и всегда были тесно связаны с развитием систем автоматизированной обработки информации. Создание баз данных после появления реляционного подхода превратилось из искусства в науку, но, как показала практика последних лет, все же окончательно его не исключило. Тем не менее сейчас это вполне сложившаяся дисциплина (хотя являющаяся скорее инженерной, чем чисто научной), основанная на достаточно формализованных подходах и включающая широкий спектр приемов и методов создания баз данных.

Как отмечается в [4], базам свойственна «перманентность» данных. Соответственно, назначение систем управления базами данных — обеспечение в течение длительного времени их сохранности, а также возможностей выборки и актуализации. Данные существуют всегда, пока есть потребность в их использовании[1], хотя характер использования, как и пути извлечения практической пользы, могут быть самыми разными: от оперативной актуализации значений до уничтожения данных, от их использования для совершенствования сложных систем управления до формирования «чемоданов компромата».

Базы данных в стремительно, а в какой-то степени и сумбурно развивающихся информационных технологиях — это сравнительно консервативное направление, где СУБД и сами базы представляют собой «долговременные сооружения». Элементная база ЭВМ и парадигмы программирования меняются быстрее, чем хранимые данные

[1] Правильнее было бы говорить, что данные создаются, но создаются не ради них самих, а для того, чтобы в дальнейшем они были использованы в каком-то процессе.

теряют актуальность. В таких условиях, в отличие от прикладных программистов, создатели баз данных (от разработчиков СУБД до администраторов БД) должны постоянно помнить о проблеме «наследственности» — о том, как интегрировать в создаваемую систему наследуемые данные, находящиеся под управлением устаревшей СУБД, и о том, как построить систему, чтобы вновь создаваемые данные могли быть, в свою очередь, наследованы следующим поколением систем и разработчиков.

Достаточно консервативны и концепции баз данных. Эта консервативность не только следствие свойства «долговечности», но и того факта, что базы вторичны по отношению к описываемым ими реальным процессам и объектам, достаточно стабильным и типичным. Кроме того, модели данных строились в значительной степени «по аналогии» с организационными и технологическими структурами.

Широкое использование баз данных различными категориями пользователей привело, с одной стороны, к созданию интерфейсов, требующих минимум времени на освоение средств управления системой, а с другой — к построению мощных, гибких СУБД, имеющих, в числе прочего, развитые средства защиты данных от случайного или преднамеренного разрушения. Появились и средства автоматизации разработки, позволяющие создать базу данных любому пользователю, даже не владеющему основами теории БД.

Но, как было отмечено ранее, база данных — это важная, но не основная функциональная, а обеспечивающая — информационная составляющая некоторой обычно достаточно крупной *человеко-машинной* системы. И здесь интересно обратить внимание на принципиальное отличие в развитии способностей взаимодействующих субъектов (человек—машина). Разделение информации на *табличную* (*числовую*), *текстовую* и *графическую* отражает последовательность, в которой эти виды информации «осваивались» компьютерами. Первые языки программирования были рассчитаны исключительно на обработку числовой информации (Fortran, Algol). Первыми появляются и табличные базы данных, также преимущественно рассчитанные на обработку числовых таблиц (файлов). Затем — текстовые файлы и текстовые БД (автоматизированные информационно-поисковые системы с библиографическими и полнотекстовыми базами). Наконец, с существенным повышением быстродействия и емкости памяти компьютеров, объектами обработки становятся графика и мультимедиа. Эта последовательность прямо противоположна той, в которой данные виды информации осваивает человек. Действительно, снача-

ла он знакомится с графическими образами (птички, цветочки и бабочки на шкафчиках в детском саду), затем — учится читать и писать, а только потом осваивает таблицу умножения.

Создание практически полезной «серьезной» базы данных в равной степени зависит как от «фундаментальности» знаний разработчика в области концепций и технологий СУБД, так и от степени понимания им сегодняшних и будущих прикладных задач пользователя, т. е. не только от адекватности применения тех или иных типовых или оригинальных решений, но и от качества представления (описания) этих решений, с той или иной степенью успешности позволяющих использовать, сопровождать и развивать систему после разработчика.

Кроме того, возможности накапливать и оперативно обрабатывать большие объемы информации, характеризующие деятельность предприятий за достаточно длительные периоды и в различных аспектах, дали новый импульс к развитию аналитических систем. Такого рода *системы поддержки принятия решений* обычно используются для оценки и выбора альтернативных решений, прогнозирования, идентификации объектов и состояний и т. д. Однако, поскольку для получения необходимых данных в этих случаях нужно использовать сложные SQL-запросы или специализированные процедуры и при этом обрабатывать большие объемы записей, это может приводить к сознательному отказу от классических нормализованных схем, так как чем выше степень нормализации, тем больше число операций соединения отношений и, соответственно, больше времени необходимо для получения конечного результата.

Базы данных — это уже достаточно хорошо проработанная научная дисциплина. Существует множество, в том числе и фундаментальных, работ и учебников (на материал которых авторы опирались при подготовке этого учебного пособия и убедительно рекомендуют их тем, кто серьезно интересуется проблематикой баз данных), среди которых необходимо выделить такие монографии, как «Организация баз данных в вычислительных системах» Дж. Мартина, «Введение в системы баз данных» К. Дейта, «Алгоритмы и структуры данных» Н. Вирта, «SQL» Дж. Гроффа и П. Вайнберга.

В своей работе авторы руководствовались и тем, что материал должен не только в компактной и наглядной форме представлять существо конкретной темы, но и подвести читателя к пониманию обоснованности (или условности) того или иного решения. Авторы сознательно избегали описаний языков и технологий, применяемых в конкретных системах, предполагая, что полноценное освоение материала

курса связано с практикой и, соответственно, с неизбежным изучением конкретных подходов, языков и технологий, свойственных выбранной системе, и изложенных в специальных пособиях, учебниках и руководствах.

Учебное пособие предназначено для студентов вузов, обучающихся по направлению 230700 «Прикладная информатика», для учащихся техникумов по специальности «Программное обеспечение вычислительной техники и автоматизированных систем». Пособие обеспечивает формирование следующих профессиональных компетенций[1] бакалавров (Б) и магистров (М):

- способность анализировать при решении профессиональных задач социально-экономические проблемы и процессы с применением методов системного анализа и математического моделирования (Б1);
- способность использовать основные законы естественнонаучных дисциплин в профессиональной деятельности и эксплуатировать современное электронное оборудование и информационно-коммуникационные технологии (Б2);
- способность применять системный подход и математические методы в формализации решения прикладных задач (Б3);
- способность осуществлять и обосновывать выбор проектных решений по видам обеспечения информационных систем (Б4);
- способность проводить обследование организаций, выявлять информационные потребности пользователей, формировать требования к информационной системе, участвовать в реинжиниринге прикладных и информационных процессов (Б5);
- способность применять методы анализа прикладной области на концептуальном, логическом, математическом и алгоритмическом уровнях (Б6);
- способность моделировать и проектировать структуры данных и знаний, прикладные и информационные процессы (Б7);
- способность оценивать и выбирать современные операционные среды и информационно-коммуникационные технологии для информатизации и автоматизации решения прикладных задач и создания ИС (Б8);
- способность проектировать и администрировать базы данных (Б9);

[1] Ниже при краткой характеристике содержания глав, будут приведены индексы соответствующих компетенций.

- способность ставить и решать прикладные задачи с использованием современных информационно-коммуникационных технологий (Б10);
- способность моделировать процессы управления и познания (М1);
- способность проектировать информационные системы с использованием современных инструментальных средств (М2);
- способность моделировать, проектировать и реализовывать системы баз данных и знаний (М3);
- умение владеть основными методами, способами и средствами получения, хранения, переработки информации, навыками работы с компьютером и информационными сетями как средством управления информацией (М4).

В целом пособие ориентировано на развитие и таких общепрофессиональных компетенций, как способность понимать роль и значение информации и информационных технологий в развитии современного общества и научного знания, а также способность использовать, обобщать и анализировать информацию, ставить цели и находить пути их достижения в условиях формирования и развития информационного общества. Организация материала пособия ориентирована на формирование способности самостоятельно приобретать и использовать в практической деятельности новые знания и умения, стремиться к саморазвитию.

Материал пособия, представленный в тринадцати главах и приложении, условно можно отнести к следующим разделам:

- введение в машинную обработку данных и структуры данных;
- введение в модели предметных областей и модели данных;
- общесистемные основы и технологии проектирования и реализации баз данных;
- язык управления данными;
- основы организации и технологии доступа к данным.

В первой главе определены основные понятия, относящиеся к базам и банкам данных, приведена классификация компонентов систем управления данными, определены их назначение и основные функции (Б4, Б10). Рассмотрен важнейший вопрос семантики баз данных в контексте информационных систем и определено соотношение понятий «информация» и «данные» (М4).

Во второй главе обсуждаются основы фактографических БД, в частности формализованного представления информации. Изложены типовые подходы к идентификации объектов и дана типология запро-

сов атрибутивного поиска описаний объектов. Определены различия между подходами, используемыми в фактографических и документальных базах данных. Вводится понятие модели данных (Б5). Приводятся основы реляционной алгебры и реляционного исчисления (М3).

В третьей главе представлены базовые технологии машинной обработки данных и рассмотрены ключевые моменты, определяющие эффективность процессов управления данными. Приведены характеристические черты систем управления данными разных поколений. Примерные схемы управления данными в файловой системе ОС и СУБД дают наглядное представление о принципиальных различиях организации процессов и разделении функций между компонентами (Б2).

Главы 4, 5, 6 посвящены проблемам моделирования баз данных. Определяются стадии проектирования и объекты моделирования. Обсуждаются отличия подходов к моделированию предметных областей. Подробно рассматривается содержание этапов проектирования и типы моделей (Б1, Б6, Б7, М3). В пятой главе подробно рассматривается концептуальное моделирование, включая модель «сущность—связь», методологии IDEF и UML (Б3, М1). В шестой главе представлены логические модели баз данных, включая реляционные, постреляционные, объектные и др. (Б3, М3). В главе 7 описывается пример проектирования реляционной базы данных, включая технологию нормализации отношений.

Главы 8 и 9 посвящены языковым средствам управления реляционными БД (Б7). Глава 9 содержит функционально полное описание SQL, который является стандартным языком для работы с реляционными базами данных (Б8, М3). Возможности использования операторов языка рассматриваются на серии примеров, иллюстрирующих этапы создания и использования базы данных, описание проектирования которой приведено в предыдущих главах.

В главе 10 приведены типовые модели физической организации данных, акцентирующие внимание на различиях в вариантах структур и связей (Б9). Рассматриваются схемы организации данных для линейных, иерархических и сетевых структур. Обсуждаются архитектуры организации данных на уровне файловых компонентов. Материал этой главы является ключевым для понимания существа внутримашинной обработки данных и, соответственно, путей построения высокоэффективных систем БД (М4).

В главе 11 представлены модели и технологии распределенной обработки данных. Проведен сравнительный анализ базовых архитек-

тур, отражающих характер распределения данных и процессов между компонентами распределенной системы. Рассмотрены типовые технологии и средства доступа к данным, распределенным в узлах вычислительной сети (Б7, Б8, М2, М4).

Глава 12 посвящена понятию «транзакция», рассматриваемому как основа технологии параллельной обработки данных. Обсуждаются модели транзакций и способы управления транзакциями (Б7, Б8, М3, М4).

Глава 13 знакомит с основными процессами управления базами данных в СУБД. Здесь затрагиваются вопросы физического планирования БД, организации управления доступом пользователей к объектам БД, программирования процессов управления обработкой данных (представления, хранимые процедуры, триггеры), а также управления репликациями и резервным копированием (Б9, М3).

Приложение содержит примерное описание физических структур реальных СУБД (М3). Сравнительный анализ этих примеров позволит прилежному читателю уяснить различия и, что нам кажется важнее, сходство решений, а также практически оценить роль моделей, которым уделялось так много внимания в большинстве глав.

Данное пособие написано в предположении, что читатели владеют математическими основами, а также знакомы с современными языками программирования.

Книга также может рассматриваться как введение в проблематику теории и практики информационных систем, основанных на базах данных. Для заинтересованного читателя материал книги должен стать отправной точкой для освоения таких дисциплин, как «Информационные системы», «Проектирование информационных систем», «Мировые информационные ресурсы», «Распределенные базы данных».

Глава 1
ВВЕДЕНИЕ В БАЗЫ И БАНКИ ДАННЫХ

1.1. Понятие базы и банка данных

Развитие вычислительной техники и появление емких внешних запоминающих устройств прямого доступа предопределило интенсивное развитие автоматических и автоматизированных систем разного назначения и масштаба, в первую очередь заметное в области бизнес-приложений. Такие системы работают с большими объемами информации, которая обычно имеет достаточно сложную структуру, требует оперативности в обработке, часто обновляется и в то же время требует длительного хранения. Примерами таких систем являются автоматизированные системы управления предприятием, банковские системы, системы резервирования и продажи билетов и т. д. (рис. 1.1).

Рис. 1.1. Примерная схема автоматизированной информационной системы

Другими направлениями, стимулировавшими развитие, стали, с одной стороны, системы управления физическими экспериментами,

обеспечивающие сверхоперативную обработку в реальном масштабе времени огромных потоков данных от датчиков, а с другой — автоматизированные библиотечные информационно-поисковые системы.

Это привело к появлению новой информационной технологии интегрированного хранения и обработки данных — *концепции баз данных,* в основе которой лежит механизм предоставления обрабатывающей программе из всех хранимых данных только тех, которые ей необходимы, и в форме, требуемой именно этой программе. При этом сама форма (структура данных и форматы полей, входящих в эту структуру) описывается на логическом, т. е. «видимом» из программы, уровне. Более того, поскольку различные программы могут по-разному «видеть» (а следовательно, и использовать) одни и те же данные, то система должна сделать «прозрачными» для программы все данные, кроме тех, которые для нее являются «своими».

Банк данных (БнД) — это система специально организованных данных, программных, языковых, организационных и технических средств, предназначенных для централизованного накопления и коллективного многоцелевого использования данных[1].

Под *базой данных (БД)* обычно понимается именованная совокупность данных, отображающая состояние объектов и их отношений в рассматриваемой предметной области. Характерной чертой баз данных является *постоянство*: данные *постоянно* накапливаются и используются; состав и структура данных, необходимых для решения тех или иных прикладных задач, обычно *постоянны* и стабильны во времени; отдельные или даже все элементы данных могут меняться — но и это есть проявление постоянства — *постоянная* актуальность.

Система управления базами данных (СУБД) — это совокупность языковых и программных средств, предназначенных для создания, ведения и совместного использования БД многими пользователями.

Иногда в составе банка данных выделяют *архивы*. Основанием для этого является особый режим использования данных, когда только часть данных находится под оперативным управлением СУБД. Все остальные данные (собственно архивы) обычно располагаются на носителях, оперативно не управляемых СУБД. Одни и те же данные в разные моменты времени могут входить как в базы данных, так и в

[1] Следует отметить, что термин «банк данных» используется сравнительно редко, а некоторыми авторами признается даже архаичным. В современной, в основном переводной литературе, например [4], понятию «банк данных» соответствует понятие *системы баз данных*, хотя, по нашему мнению, «банк данных» вполне адекватное и более широкое понятие.

архивы. Банки данных могут не иметь архивов, но если они есть, то в состав банка данных может входить и система управления архивами.

Проблемы совместного использования данных и периферийных устройств компьютеров и рабочих станций быстро породили модель вычислений, основанную на концепции файлового сервера — сеть создает основу для коллективной обработки, сохраняя простоту работы с персональным компьютером, позволяет совместно использовать данные и периферию.

В этом смысле главной отличительной чертой баз данных является использование централизованной системы управления данными, причем как на уровне файлов, так и на уровне элементов данных. Централизованное хранение совместно используемых данных приводит не только к сокращению затрат на создание и поддержание данных в актуальном состоянии, но и к сокращению избыточности информации, упрощению процедур поддержания непротиворечивости и целостности данных.

Эффективное управление внешней памятью является основной функцией СУБД. Эти обычно специализированные средства настолько важны с точки зрения эффективности, что при их отсутствии система просто не сможет выполнять некоторые задачи уже потому, что их выполнение будет занимать слишком много времени. При этом ни одна из таких специализированных функций, как построение индексов, буферизация данных, организация доступа и оптимизация запросов, не является видимой для пользователя и обеспечивает независимость между логическим и физическим уровнями системы: прикладной программист не должен писать программы индексирования, распределять память на диске и т. д.

Развитие теории и практики создания информационных систем, основанных на концепции баз данных, создание унифицированных методов и средств организации и поиска данных позволяют хранить и обрабатывать информацию о все более сложных объектах и их взаимосвязях, обеспечивая многоаспектные информационные потребности различных пользователей. Основные требования, предъявляемые к банкам данных, можно сформулировать следующим образом [12].

Многократное использование данных: пользователи должны иметь возможность использовать данные различным образом.

Простота: пользователи должны иметь возможность легко узнать и понять, какие данные имеются в их распоряжении.

Легкость использования: пользователи должны иметь возможность осуществлять (процедурно) простой доступ к данным, при этом все

сложности доступа к данным должны быть скрыты в самой системе управления базами данных.

Гибкость использования: обращение к данным или их поиск должны осуществляться с помощью различных методов доступа.

Быстрая обработка запросов на данные: запросы на данные, в том числе незапланированные, должны обрабатываться с помощью высокоуровневого языка запросов, а не только прикладными программами, написанными с целью обработки конкретных запросов (разработка таких программ в каждом конкретном случае связана с большими затратами времени). Пользователь должен иметь возможность кратко выразить нетривиальные запросы (в нескольких словах или несколькими нажатиями клавиш мыши). Это означает, что средство формулирования должно быть достаточно «декларативным», т. е. упор должен быть сделан на «что», а не на «как». Кроме того, средство обработки запросов не должно зависеть от приложения, т. е. оно должно работать с любой возможной базой данных.

Язык взаимодействия конечных пользователей с системой должен обеспечивать конечным пользователям возможность получения данных без использования прикладных программ.

База данных — это основа для будущего наращивания прикладных программ: базы данных должны обеспечивать возможность быстрой и дешевой разработки новых приложений.

Сохранение затрат умственного труда: существующие программы и логические структуры данных (на создание которых обычно затрачивается много человеко-лет) не должны переделываться при внесении изменений в базу данных.

Наличие интерфейса прикладного программирования: прикладные программы должны иметь возможность просто и эффективно выполнять запросы на данные; программы должны быть изолированы от расположения файлов и способов адресации данных.

Распределенная обработка данных: система должна функционировать в условиях вычислительных сетей и обеспечивать эффективный доступ пользователей к любым данным распределенной БД, размещенным в любой точке сети.

Адаптивность и расширяемость: база данных должна быть настраиваемой, причем настройка не должна вызывать перезаписи прикладных программ. Кроме того, поставляемый с СУБД набор предопределенных типов данных должен быть расширяемым — в системе должны иметься средства для определения новых типов и не должно

быть различий в использовании системных и определенных пользователем типов.

Контроль за целостностью данных: система должна осуществлять контроль ошибок в данных и выполнять проверку взаимного логического соответствия данных.

Восстановление данных после сбоев: автоматическое восстановление без потери данных транзакции. В случае аппаратных или программных сбоев система должна возвращаться к некоторому согласованному состоянию данных.

Вспомогательные средства должны позволять разработчику или администратору базы данных предсказать и оптимизировать производительность системы.

Автоматическая реорганизация и перемещение: система должна обеспечивать возможность перемещения данных или автоматическую реорганизацию физической структуры.

1.2. Компоненты банка данных

Определение банка данных предполагает, что с функционально-организационной точки зрения банк данных является сложной человеко-машинной системой, включающей в себя все подсистемы, необходимые для надежного, эффективного и продолжительного во времени функционирования.

В структуре банка данных выделяют следующие компоненты (подсистемы):

- информационная база;
- лингвистические средства;
- программные средства;
- технические средства;
- организационно-административные подсистемы и нормативно-методическое обеспечение.

1.2.1. Информационная база

Данные, отражающие состояние определенной предметной области (ПрО) и используемые информационной системой, принято называть *информационной базой*. Информационная база состоит из двух компонентов: 1) коллекции записей собственно данных; 2) описания этих данных — метаданных.

Данные отделены от описаний, но в то же время данные не могут использоваться без обращения к соответствующим описаниям.

Уже из определения базы данных и приведенных ранее основных требований следует, что данные могут использоваться (т. е. представляться) по-разному. С одной стороны, разные прикладные задачи требуют разных наборов данных, в совокупности обеспечивающих функциональную полноту информации, а с другой — они должны быть различны для различных категорий субъектов (разработчиков или пользователей). Также должны быть различными и способы описания самих данных, их природы, формы хранения, условий взаимной непротиворечивости.

В литературе по базам данных упоминаются три уровня представления данных — концептуальный, внешний (логический) и внутренний (физический).

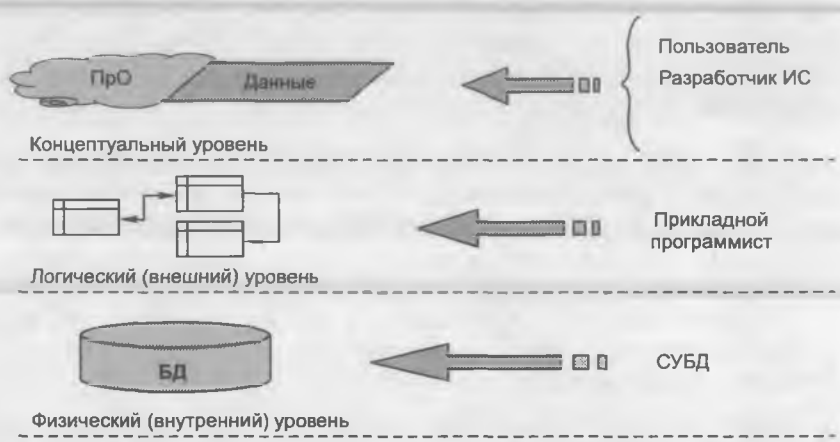


Рис. 1.2. Уровни представления данных

Эти уровни представлений введены исходя из различного рассмотрения БД. Например, прикладному программисту требуются не все данные БД, а только некоторая их часть, используемая в его программе. Внешний уровень представления обеспечивает именно эту форму обмена данными.

Внутренний уровень — представление БД, которое полностью определяет необходимые условия для организации хранения данных на внешних запоминающих устройствах.

Описание БД на концептуальном уровне представляет собой обобщенный взгляд на данные с позиций предметной области (разра-

ботчика приложений, пользователя или внешней информационной системы).

Внешний уровень представления данных не затрагивает физической организации (размещения) данных во внешней памяти, поэтому его называют иногда логическим уровнем. Соответственно внутренний уровень называют физическим уровнем.

1.2.2. Лингвистические средства

Многоуровневое представление БД предполагает соответствующие описания данных на каждом уровне и согласование одних и тех же данных на разных уровнях. С этой целью в состав СУБД включаются специальные языки для описания представлений внутреннего и внешнего уровней. Кроме того, СУБД должна включать в себя язык манипулирования данными (ЯМД). Желательно также наличие тех или иных дополнительных сервисных средств, например средств генерации отчетов.

Работа с базами данных предполагает несколько этапов:

- описание БД;
- описание частей БД, необходимых для конкретных приложений (задач, групп задач);
- программирование задач или описание запросов в соответствии с правилами конкретного языка и использованием языковых конструкций для обращения к БД;
- загрузка БД и т. д.

Для выражения обобщенного взгляда на данные применяют *язык описания данных (ЯОД)* внутреннего уровня, включаемый в состав СУБД[1]. Описание представляет собой модель данных на уровне структур, из которых образуется БД.

ЯОД позволяет определять схемы базы данных, характеристики хранимых и виртуальных данных и параметры организации их хранения в памяти и может включать в себя средства поддержки целостности базы данных, ограничения доступа, секретности.

ЯМД обычно включает в себя средства запросов к базе данных и ведения базы данных (добавление, удаление, обновление данных,

[1] Отсюда следует, что одна и та же БД может описываться по-разному на ЯОД различных СУБД.

создание и уничтожение отдельных структур и БД в целом, изменение БД на структурном уровне и т. п.).

Функциональные характеристики языков отражают возможности описания данных, средств представления запроса, обновления, поддержки целостности и секретности, включения в языки программирования, управления форматом ответов, средств запроса к словарю данных БД и т. д.

Качественные характеристики языков запросов могут определяться такими свойствами, как полнота, селективная мощность, простота изучения и использования, степень процедурности и модульности, унифицированность, производительность и эффективность. Рассмотрим некоторые из этих понятий.

Селективная мощность языков запросов характеризует возможность выбора данных по разным критериям. Данное понятие плохо поддается формализации, можно сказать, что язык с большей селективной мощностью позволяет сформулировать большинство запросов так, что ответ на них содержит меньше ненужных данных. Языки, обладающие малой селективной мощностью, в общем случае уже требуют привлечения дополнительных средств для анализа ответов на запросы (например, оценки пользователя).

Простота изучения является во многом субъективной оценкой и может быть в некоторой мере охарактеризована степенью его близости к естественному языку, требуемым для его освоения временем и необходимым уровнем подготовки пользователя.

Высокий *уровень процедурности*, свойственный реляционным языкам, определяется присущими реляционной модели свойствами, в частности полным отделением логической структуры данных от структур хранения и стратегий доступа. Снижение уровня процедурности увеличивает свободу в выборе способов реализации языка, что позволяет осуществить его реализацию более оптимальным способом. Однако необходимо отметить, что меньшая степень процедурности еще не означает автоматически меньшую сложность написания запросов. Некоторые сложные запросы можно более просто сформулировать в виде алгоритма поиска ответа, в то время как его формулировка в декларативном виде может оказаться достаточно трудной.

Модульность построения языка характеризует возможность существования нескольких уровней языка и зависит от специфических свойств математической теории, лежащей в его основе. Минимальный уровень языка, обычно легко понимаемый пользователем, бывает достаточным для формулирования большинства запросов, и лишь

формулировка сложных запросов может потребовать использования всех выразительных средств языка, о существовании которых пользователи начального уровня могут и не знать. Языки, не обладающие модульностью, требуют от пользователя знания почти всего объема средств языка, что усложняет процесс их изучения.

Наиболее распространенным языком для работы с базами данных является SQL (Structured Query Language), в своих последних реализациях предоставляющий не только средства для спецификации и обработки запросов на выборку данных, но также и функции по созданию, обновлению, управлению доступом и т. д.

По существу SQL уже соединяет в себе язык описания данных и язык манипулирования данными. Он не является полноценным языком программирования, и в случае его использования для организации доступа к БД из прикладных программ SQL-выражения встраиваются в конструкции базового языка.

Являясь внутренним языком баз данных, SQL естественно отражает особенности конкретной СУБД. Сегодня это единственный стандартизованный язык фактографических баз данных, достаточно мощный и в то же время простой для понимания и использования. Сочетание этих факторов вместе с поддержкой ведущих производителей, таких как IBM и Microsoft, привели не только к широкому его распространению, но и совершенствованию. Сегодня, благодаря независимости от конкретных СУБД и межплатформенной переносимости, SQL стал языком распределенных баз данных и языком шлюзов, позволяющим совместно использовать СУБД разного типа.

1.2.3. Программные средства

Обработка данных и управление этой обработкой в вычислительной среде, а также взаимодействие с операционной системой и прикладными программами осуществляется комплексом программных средств, взаимосвязь которых иллюстрируется рис. 1.3. В составе комплекса обычно выделяют следующие компоненты:

- *ядро*, обеспечивающее управление данными во внешней и оперативной памяти, а также протоколирование изменений;
- *процессор языка базы данных*, обеспечивающий обработку (трансляцию или компиляцию) и оптимизацию запросов на выборку и изменение данных;

- *подсистему (библиотеку) поддержки программных вызовов*, которая обслуживает прикладные программы управления данными, взаимодействующие с СУБД через средства пользовательского интерфейса;
- *сервисные программы* (системные и внешние утилиты), обеспечивающие настройку СУБД, восстановление после сбоев и ряд дополнительных возможностей обслуживания.

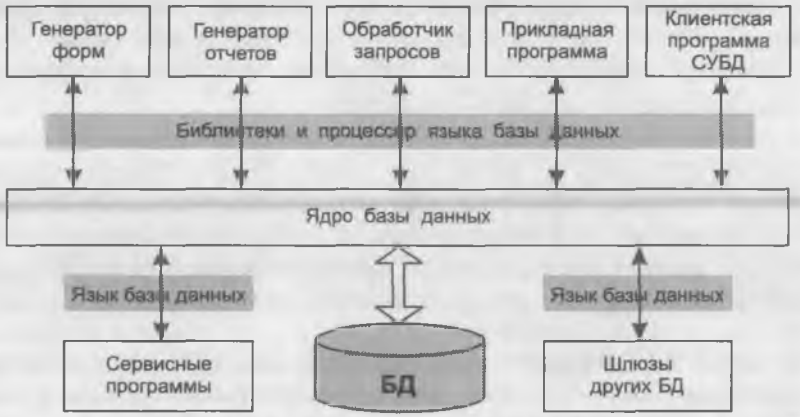


Рис. 1.3. Программные средства СУБД

Большинство СУБД работают в среде операционной системы и тесно с ней связаны. Многопользовательские приложения, обработка распределенных запросов, защита данных требуют эффективно использовать ресурсы, управление которыми обычно является функцией ОС. Использование многопроцессорных систем и мультипоточных технологий обработки данных позволяет эффективно обслуживать параллельно выполняемые запросы, но требует координации использования ресурсов между ОС и СУБД. Соответственно, управление доступом и обеспечение защиты также обычно интегрируются с соответствующими средствами операционной системы.

Именно централизованное управление данными обеспечивает:

- сокращение избыточности в хранимых данных;
- совместное использование хранимых данных;
- стандартизацию представления данных, упрощающую эксплуатацию БД;
- разграничение доступа к данным;

- целостность данных, обеспечиваемую процедурами, предотвращающими включение в БД неверных данных, и ее восстановление после отказов системы.

1.2.4. Технические средства

Сегодня большинство банков данных создается и функционирует на основе универсальных вычислительных машин[1]. Однако для больших баз данных, функционирующих в промышленном режиме, обеспечение эффективной и бесперебойной работы должно основываться на использовании адекватных аппаратных средств.

Устройства ввода-вывода и накопители внешней памяти — традиционно узкое место любой базы данных. Объем и быстродействие накопителей являются, очевидно, важными параметрами. Однако столь же значима и отказоустойчивость. Здесь следует отметить необходимость согласованных решений при распределении ролей между аппаратными и программными компонентами управления операциями ввода-вывода. Например, наличие буферной памяти в накопителе, ускоряющей ввод-вывод (аппаратное кэширование) при сбоях системы во время выполнения операции записи в БД может привести к потере данных: переданные для записи данные еще будут находиться в буфере, а так как СУБД отметит операцию записи как уже завершившуюся, откат для восстановления данных станет невозможен.

Для повышения надежности хранения часто используют специализированные дисковые подсистемы — RAID (Redundant Array of Inexpensive Disk). Один логический RAID-диск — это несколько физических дисков, объединенных в одно устройство, управляемое специализированным контроллером, что позволяет распределять основные и системные данные между несколькими носителями (дисками), в том числе дублировать данные. Таким образом, в случае повреждения одного из дисков можно оперативно восстановить потерянные данные.

---

[1] Здесь следует упомянуть достаточно интенсивно развивавшееся в 1980—1990-е гг. направление создания машин баз данных — аппаратной реализации «нечисловой» обработки, в том числе параллельной и конвейерной обработки, ассоциативных процессоров и памяти [17].

Сегодня для реализации промышленных БД используются специализированные *серверы баз данных* — машины с повышенной отказоустойчивостью, высокопроизводительными подсистемами ввода-вывода и развитой периферией.

Не менее значима роль центрального процессора. Многие промышленные СУБД поддерживают многопроцессорную обработку запросов. Теоретически использование еще одного процессора позволит ускорить обработку. Однако на практике многопроцессорные системы требуют повышенного внимания при приобретении оборудования: надежно работают только сертифицированные системы, использующие соответствующие периферийные устройства.

Для распределенных и удаленно используемых баз данных также важно сетевое окружение: связное оборудование и сетевые протоколы. Здесь важны не только показатели быстродействия, но и поддерживаемые ими возможности обеспечения безопасности.

1.2.5. Организационно-административные подсистемы

Организационно-методические средства не являются техническим компонентом системы, однако трудно рассчитывать на устойчивое и долговременное функционирование банка данных, если будут отсутствовать необходимые методические и инструктивные материалы, регламентирующие работу пользователей, различных по своему статусу и уровню подготовленности.

1.3. Пользователи баз данных

В информационных системах, создаваемых на основе СУБД, способы организации данных и методы доступа к ним перестали играть решающую роль, поскольку оказались скрытыми внутри СУБД. Массовый, так называемый *конечный пользователь*, как правило, имеет дело только с внешним интерфейсом, поддерживаемым СУБД.

Эти преимущества, как уже понятно, не могут быть реализованы путем механического объединения данных в БД. Предполагается, что в системе обязательно существует специальное должностное лицо (группа лиц) — *администратор базы данных (АБД),* который несет ответственность за проектирование и общее управление базой данных. АБД определяет информационное содержание БД. С этой целью он идентифицирует объекты БД и моделирует базу, используя язык описания данных. Получаемая модель служит в дальнейшем справочным документом для администраторов приложений и пользователей. Ад-

министратор решает также все вопросы, связанные с размещением БД в памяти, выбором стратегии и ограничений доступа к данным. В функции АБД входят также организация загрузки, ведения и восстановления БД и многие другие действия, которые не могут быть полностью формализованы и автоматизированы.

Администратор приложений (или, если таковой специально не выделяется, администратор БД) определяет для приложений подмодели данных. Тем самым разные приложения обеспечиваются собственным «взглядом», но не на всю БД, а только на требуемую для конкретного приложения («видимую») ее часть. Вся остальная часть БД для данного приложения будет «прозрачна».

Прикладные программисты имеют, как правило, в своем распоряжении один или несколько языков программирования, с помощью которых генерируются прикладные программы.

1.4. Типология баз данных

Классификация баз и банков данных может быть произведена по разным признакам (относящимся к разным компонентам и сторонам функционирования банков данных, среди которых выделяют, например, следующие.

По *форме представляемой информации* можно выделить фактографические, документальные, мультимедийные, в той или иной степени соответствующие цифровой, символьной и другим (нецифровой и несимвольной) формам представления информации в вычислительной среде. К последним можно отнести картографические, видео-, аудио-, графические и другие БД.

По *типу хранимой (не мультимедийной) информации* можно выделить фактографические, документальные, лексикографические БД. Лексикографические базы — это классификаторы, кодификаторы, словари основ слов, тезаурусы, рубрикаторы и т. д., которые обычно используются в качестве справочных совместно с документальными или фактографическими БД. Документальные базы подразделяются по уровню представления информации на полнотекстовые (так называемые «первичные» документы) и библиографическо-реферативные («вторичные» документы, отражающие на адресном и содержательном уровнях первичный документ).

По *типу используемой модели данных* выделяют три классических класса БД: иерархические, сетевые, реляционные. Развитие техноло-

гий обработки данных привело к появлению постреляционных, объектно ориентированных, многомерных БД, которые в той или иной степени соответствуют трем упомянутым классическим моделям.

По *топологии хранения* данных различают локальные и распределенные БД.

По *типологии доступа и характеру использования* хранимой информации БД могут быть разделены на специализированные и интегрированные[1].

По *функциональному назначению* (характеру решаемых с помощью БД задач и, соответственно, характеру использования данных) можно выделить операционные и справочно-информационные. К последним можно отнести ретроспективные БД (электронные каталоги библиотек, БД статистической информации и т. д.), которые используются для информационной поддержки основной деятельности и не предполагают внесения изменений в уже существующие записи, например по результатам этой деятельности. Операционные БД предназначены для управления различными технологическими процессами. В этом случае данные не только извлекаются из БД, но и изменяются (добавляются), в том числе в результате этого использования.

По *сфере возможного применения* можно различать универсальные и специализированные (или проблемно ориентированные) системы.

По *степени доступности* можно выделить БД общедоступные и с ограниченным доступом пользователей. В последнем случае говорят об управляемом доступе, индивидуально определяющем не только набор доступных данных, но и характер операций, которые доступны пользователю.

Следует отметить, что представленная классификация не является полной и исчерпывающей. Она в большей степени отражает исторически сложившееся состояние дел в сфере деятельности, связанной с разработкой и применением баз данных.

1.4.1. Фактографические и документальные БД

Главное различие фактографических и документальных БД проявляется в структуре единицы хранения информации.

---

[1] В последнем случае правильнее говорить об интегрированных информационных системах, объединяющих в общей среде разнородные данные, хранимые, возможно, в разнотипных базах, но используемые для решения одной прикладной задачи.

Под *единицей хранения информации* будем понимать совокупность данных, которая с точки зрения информационной системы представляет собой единое целое. Единица хранения определяет свойства целостности и непротиворечивости данных.

С точки зрения структуры единицы хранения принято различать хорошо структурированные и слабо структурированные данные.

Хорошо структурированные данные — это данные, в которых каждую единицу хранения информации можно представить в качестве конечного набора атрибутов. При этом каждый из них будет принимать точно определенное значение.

Слабо структурированные данные — это данные, в которых каждую единицу хранения также представляют конечным числом атрибутов, но значение атрибута априорно точно не определено, например зависит от контекста использования, а сам атрибут, в свою очередь, может иметь сложную структуру.

Фактографические БД ориентированы на хранение хорошо структурированных данных. Единицей хранения в таких БД служит описание «факта», задаваемое конечным четко определенным множеством характеристических свойств (атрибутов).

При построении концептуальной модели таких БД предметная область (ПрО) естественным путем декомпозируется на объекты и связи между ними. Каждое характеристическое свойство объекта имеет атомарное значение, которое не зависит от контекста использования.

Документальные БД предназначены для хранения слабо структурированных данных. Единицей хранения при этом является документ, заданный конечным, но, возможно, не фиксированным набором полей произвольной длины.

При построении документальных БД обычно ПрО представляется как совокупность в общем случае не взаимодействующих объектов. Набор характеристических свойств объекта конечен, но не фиксирован. Значение характеристического свойства может быть множественным и может зависеть от контекста использования.

С точки зрения методов и алгоритмов поиска фактографические БД рассматривают как информационное обеспечение *поиска данных*, а документальные БД — как информационное обеспечение *поиска информации*.

Соответственно, различие в основных функциональных требованиях к данным приводит к различию фактографических и документальных баз на уровне структур данных.

1.4.2. Операционные и справочно-информационные БД. Хранилища данных

Операционные БД (или БД оперативной информации) являются основой так называемых *OLTP-приложений* (On-Line Transactions Processing). Типичными примерами таких приложений могут служить системы заказа и покупки билетов, банковские операционные системы и т. п. Основное назначение OLTP-приложений — обеспечить одновременное выполнение большого количества операций манипулирования данными, объединенных в *транзакции*[1] (например, «продать пассажиру билет на заданные дату, поезд и место»).

Запросы к БД от OLTP-приложений в основном состоят из команд вставки, удаления, модификации данных. Запросы на выборку носят фиксированный, регламентированный характер и формулируются еще на этапе проектирования. Тем самым критичными для OLTP-приложений являются скорость и надежность выполнения операций обновления данных.

Справочно-информационные БД являются основой как документальных информационных систем (БД ретроспективной информации), ориентированных на задачи информационного поиска, так и *OLAP-приложений* (On-Line Analitical Processing). OLAP — оперативная аналитическая обработка данных — обобщенный термин, характеризующий принципы построения *систем поддержки принятия решений* (DSS — Decision Support System), а также *систем интеллектуального анализа данных* (Data mining).

Данные в справочно-информационных БД обычно накапливаются и практически никогда не удаляются, а добавление новых данных происходит большими (накопленными) порциями и относительно редко. Запросы имеют нерегламентированный характер и потому не могут быть сформулированы окончательно на этапе проектирования. Скорость выполнения запросов обычно не критична.

При этом именно операционные БД обычно являются основным источником пополнения справочно-информационных, используемых в задачах поддержки принятия решений и анализа данных. Более того, справочно-информационные БД могут пополняться данными, поступающими из различных источников, причем в разных источниках одни и те же данные могут иметь разный формат представления

[1] Подробнее про транзакции см. в главе 11.

(т. е. данные перед загрузкой должны проходить различные процедуры преобразования и «очистки»).

Для решения задачи интегрирования и хранения данных для аналитической обработки, поступающих из разных источников, была предложена концепция хранилища данных.

В [10] приводится следующее определение.

Хранилище данных — предметно-ориентированный, интегрированный, привязанный ко времени и неизменяемый набор данных, предназначенный для поддержки принятия решений.

Предметная ориентированность хранилища характеризует направленность на накопление и хранение данных о статичных предметах (объектах, фактах), а не о динамике функционирования предметной области (например, в хранилище накапливаются характеристики выполненных заказов, а не данные о ходе выполнения работ по заказам).

Интегрированность характеризует возможность накапливать оперативные данные из разных источников, в том числе и в случае несогласованности представления одних и тех же данных.

Привязка ко времени данных в хранилище предполагает обязательное указание момента времени или временного интервала для каждого поступающего в хранилище факта.

Таблица 1.1. **Сравнительная характеристика операционных БД и хранилищ данных**

	Операционная БД	Хранилище данных
Характер хранимых данных	Содержит текущие данные	Содержит исторически накапливаемые данные
Предварительное агрегирование данных	Не происходит	Возможно
Динамика данных	Данные постоянно обновляются	Данные в основном являются статическими
Характер запросов	Регламентированный	Нерегламентированный
Интенсивность обработки	Высокая	Средняя и низкая
Назначение	Обработка транзакций и поддержка принятия оперативных решений	Анализ данных и поддержка принятия стратегических решений

Неизменяемость данных в хранилище предполагает только пополнение за счет данных из оперативных систем обработки. При этом новые данные никогда не заменяют и не изменяют уже имеющиеся в хранилище, а старые данные не подлежат изменению.

Конечная цель создания хранилища данных — интеграция данных в едином пространстве, обеспечивающая поддержку управления данными и их анализа. Сравнительную характеристику операционных БД и хранилищ данных иллюстрирует табл. 1.1.

1.4.3. Типология баз данных с точки зрения информационных процессов

БД могут рассматриваться на различных уровнях *информационных процессов:* уровне информационных технологий (ИТ), уровне информационных систем (ИС), уровне информационных ресурсов (ИР).

На уровне информационных технологий БД определяется как взаимосвязанная совокупность файлов операционной системы (ОС), содержащих данные о предметной области решаемой задачи. При этом основное внимание уделяется *физической структуре БД.*

На уровне информационных систем БД рассматривается как компонент, представляющий собой информационную модель предметной области. Здесь наиболее важной является проблема *логической структуры БД.*

При рассмотрении на уровне информационных ресурсов БД трактуется как элемент мировых ИР. Основной характеристикой здесь является *содержание БД* (хотя структуры данных также важны).

Основное внимание в данном пособии будет уделяться рассмотрению БД на уровне технологии и систем, уровень ИР будет вкратце рассмотрен только в настоящей главе (п. 1.5).

Программные средства баз данных. Оболочки информационных систем (системы программирования ИС) представляют собой гибкие программные комплексы, настраиваемые на задачи пользователя. Наиболее распространенными классами данных программных средств являются *системы управления базами данных* (СУБД) и оболочки *автоматизированных информационно-поисковых систем* (АИПС).

Информационно-поисковые системы. В узком смысле под АИПС принято понимать открытый или замкнутый программный продукт, предназначенный для реализации функций (процессов) *ввода, обработки, хранения, поиска, представления данных* (организованных в за-

писи или документы, находящиеся в БД). В этом смысле часто отождествляют АИПС с АИС, и это трудно оспаривать.

Среди АИПС в узком смысле принято выделять:

- *фактографические системы* (работающие с хорошо структурированными данными), для разработки которых, как правило, используются СУБД;
- *документальные системы* (работающие со слабо структурированными данными), разработка которых часто (но не обязательно) ведется с использованием оболочки АИПС.

В более широком смысле под АИПС подразумеваются также *программные оболочки, ориентированные на разработку продуктов типа АИПС* (в узком смысле).

Системы управления базами данных и программирования АИС. Среди различных программных средств данного класса следует различать три типа:

- СУБД в чистом виде;
- СУБД с элементами систем программирования АИС;
- системы программирования АИС с элементами СУБД.

СУБД первого типа фактически включают только систему интерпретации вызовов (обращений) из пользовательской программы (call-interface) на выборку (корректировку, занесение) информации из (в) БД.

Второй тип представляет собой расширение первого в направлении создания универсальной системы разработчика АИС, включающей также специализированные языковые средства. В этом случае СУБД представляет собой совокупность специализированных программных средств, вспомогательных файлов и управляющих объектов (иногда находящихся в составе БД, реже это файлы ОС), обеспечивающих доступ пользователей к БД при соблюдении следующих существенных критериев: целостность и непротиворечивость данных, защита информации от несанкционированного доступа на чтение/обновление содержимого БД, установление и поддержание связей между зависимыми данными, удобство использования данных.

Третий тип представляют собой системы программирования, содержащие элементы как непроцедурного типа (язык запросов), так и процедурного (язык программирования). Свойства СУБД здесь также проявляются в наличии простейшего словаря данных, возможности создания модели предметной области в форме совокупности объектов, связанных между собой простейшим образом, а также в наличии средств генерации отчетов и управления доступом пользователей.

1.5. Семантика баз данных[1]

Как уже отмечалось, база данных не может рассматриваться в отрыве от назначения и особенностей ее использования для решения практических задач, причем обязательно в составе более крупных информационных или технологических автоматизированных систем. Задачи таких систем — не только планирование и управление предприятием, но и интеграция разработки и сопровождения основных и технологических объектов и процессов, диагностика, мониторинг, моделирование. Соответственно, задачи и назначение БД как системы, хранящей информацию обо всех этих составляющих, — обеспечить информационную поддержку этих процессов.

База данных — это отражение реальной предметной области, «действующая» информационная модель[2], которая, обеспечивая субъект информацией для принятия решения, позволяет в числе прочего и управлять объектами и процессами в отражаемой предметной области. Такая функциональная направленность (естественно, предполагающая достижение эффективности в первую очередь за счет использования именно БД) обусловливает и обратную зависимость: объекты, процессы и события ПрО выделяются таким образом, чтобы было возможно их представление в виде системы взаимосвязанных данных и процессов, удобных для их последующей (человеко-машинной) обработки.

В каком-то смысле базу данных можно сравнить с сообщением о состоянии предметной области, воспринимаемым некоторым субъектом, задачей которого и является преобразование объектов этой ПрО, причем в своей деятельности субъект руководствуется информацией, извлекаемой именно из этого «сообщения». Схема этого соотношения, приведенная на рис. 1.4, иллюстрирует еще и то, что система, преобразующая объект, принципиально является комплексной (состоящей по крайней мере из двух компонентов, работающих с объектами разной природы: субъект преобразования взаимодействует пре-

[1] Материал этого и следующего параграфов является не только введением в проблематику проектирования и эксплуатации баз данных, но и, может быть, несколько опережающим обобщением того, что будет представлено в дальнейших главах.

[2] Модель лишь в том смысле, что она — представление, описание на уровне данных только некоторых аспектов и только некоторой части реального мира, и поэтому не может быть тождественна реальным объектам. Но в то же время БД и сама является частью реального мира.

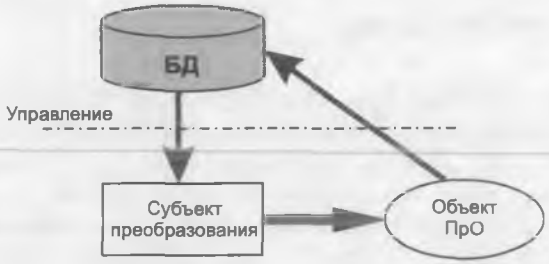


Рис. 1.4. Информационная модель преобразования

имущественно с материальными объектами, а БД — с информационными).

Для многокомпонентных систем с многоуровневым представлением семантики эффективность обработки достигается через специализированность представления объектов или процессов (а для вычислительных систем — как среды хранения информации — с *единственно возможной* **двоичной** формой представления) и, в первую очередь, путем сведения представления множества обрабатываемых (локально) объектов к однородности природы и формы их представления. Поэтому в общем случае для реализации эффективного межуровневого взаимодействия (на каждом из уровней объекты представлены в виде, наиболее адекватном функциональным средствам этого уровня) любая величина должна быть преобразована в соответствии с «контекстом» этого уровня для получения такого ее представления, которое будет значимо для воспринимающего уровня, т. е. может быть обработано средствами этого уровня.

Здесь «контекст» — это декларативное или иногда процедурное определение способа использования элементарных составляющих величины для получения значения. Например, порядок использования байтов при преобразовании вещественного числа, представленного в двоичной форме, в символьный формат.

Соотношение понятий *величина*, *контекст* и *значение* приведено на рис. 1.5. Здесь значение, получаемое в первом процессе (на первом уровне), в следующем рассматривается в свою очередь как величина, которая будет интерпретироваться в соответствии с контекстом своего процесса[1].

[1] Соотношение понятий «величина» и «значение» аналогично соотношению понятий «данные» и «информация». Информация — это значимые для приемника данные, например изменяющие его внутреннее состояние.

Рис. 1.5. Соотношение понятий «величина», «контекст» и «значение»

Таким образом, можно сказать, что значение в общем случае определяется парой <*контекст, величина*>. Причем, поскольку *контекст* и *величина* имеют разную природу, они должны быть представлены в вычислительной среде самостоятельными, скорее всего разнотипными объектами.

Такое, хотя и упрощенное, представление о БД как о средстве информационных коммуникаций позволяет тем не менее увидеть взаимосвязь вида информации (способа реализации смысла) с формой ее представления и особенностью ее использования.

В этом смысле (с точки зрения способа представления и, соответственно, восприятия) в отдельный класс можно выделить *фактографическую информацию:* такое представление реально существующих событий и явлений, когда они могут быть описаны как *факты*, задаваемые парой <*имя, значение*>, где *имя* — знак, уникально определяющий (идентифицирующий) факт в заданной предметной области и обычно не нуждающийся в явном определении или доопределении его существа, а *значение* — характеристика, задающая одно из множества возможных состояний.

Таким образом, здесь факт (его значение) задается величиной, например числовой, для параметров, измеримых физически, в том числе и логическими величинами «истина»/«ложь» для указания, свершилось событие или нет[1].

Можно сказать, что особенностью фактографической информации является практическая очевидность (минимальная неопределенность, не требующая использования сложных или нечетких процедур) идентификации и интерпретации «факта» — как его имени, так и со-

[1] И следует отметить, что такая форма в наибольшей степени соответствует машинным формам представления информации.

стояния. Таким образом, контекст в этом случае в достаточной степени определяется однозначно понимаемым объявлением о назначении базы данных и таким именованием полей данных, когда в качестве имени используется общепринятое, не зависящее от прикладных задач *имя свойства* (и таким образом определяются характеристические признаки). Такая ситуация предопределяет для пользователя возможность адекватного восприятия содержания: способ интерпретации данных в этом случае практически не может быть неоднозначным, причем для пользователя *определение способа* происходит *неявно* (не требует от него явных действий для определения и использования контекста). Это, с одной стороны, позволяет свести представление предметной области к точной теоретико-множественной модели, а с другой — обусловливает возможность непосредственного использования данных в задачах обработки (на уровне прикладных программ) для генерации новой информации без участия субъекта (человека), внешнего по отношению к машинной среде, обеспечивающего определение и использование контекста. Например, OLAP-технологии баз данных, позволяют строить на основе множества данных, количественно характеризующих состояние объектов предметной области и представленных обычно регулярными таблицами, новые значения, отражающие это состояние на ином *качественном* уровне, — интегральные показатели, диаграммы, графики и т. д.

Однако большинство задач, решаемых человеком, не могут быть сведены к «фактографическому» представлению и описываются (и, соответственно, представляются в машинной среде) средствами естественного или специализированного языков, оперирующих *лингвистическими переменными*, значение которых может зависеть не только от контекста предметной области, но также и от контекста ближайшего окружения — значения соседних переменных. Причем появление нового смысла (факта) не обязательно приводит к появлению новой переменной: новый факт представляется с помощью уже существующих переменных. Например, словесные определения философских или географических понятий.

В отличие от ранее рассмотренного фактографического представления, для вербальной формы представления факта (выражениями языка с использованием лингвистических переменных) характерно то, что для задания *имени*, *значения* и *контекста* может использоваться единый способ и средства — лингвистические переменные одного и того же языка. Например, описание весовых свойств может быть представлено несколькими, но имеющими один смысл вариантами пред-

ложений: «Чугунная заготовка весом 29 килограммов» или «Чугунная заготовка имеет свойство $m = 29$, где m — масса в килограммах».

Автоматическое приведение такого рода представлений к очевидно наилучшей для этого случая табличной форме потребовало бы применения трудно реализуемых процедур морфологического и семантического анализов. Однако, с другой стороны, выделение смысла (и генерация новой информации) обычно производится человеком, сознание которого (как среда преобразования) ориентировано именно на обработку лингвистических переменных.

Рассматривая процесс автоматизированной генерации новой информации (рис. 1.6), где в качестве источника исходных данных используются БД, нужно сказать, что отбор и обработка должны быть выделены в отдельные процессы, так как с точки зрения общей (суммарной) эффективности один из них (обычно поиск) должен быть опосредованным: оценка полезности найденной информации производится обычно человеком, так как сознание человека — внешняя по отношению к машине среда — работает со слабо структурированной информацией эффективнее машин.

Рис. 1.6. Схема процесса автоматизированного решения задач

Случаи, когда информация представляется в форме, не адекватной архитектуре фон-неймановских машин, могут быть обусловлены разными факторами. Рассмотрим следующие случаи.

1. Хорошо структурированная информация, представляемая в графическом или специальном формате. Например, структурные химические формулы, конструкторская документация и т. д. В этом случае для автоматической обработки требуются узкоспециализированные средства, что приводит к общей неунифицированности представления семантических элементов (например, графических примитивов) на уровне данных.

2. Информация, точная по содержанию, но вариантно представляемая по форме. Например, описание в текстовом виде численно задаваемых параметров изделия. Лингвистические переменные в этом случае имеют точное значение, однако построение универсальной процедуры автоматического выделения факта из текста трудоемко и потому нецелесообразно.

3. Слабо структурированная информация, обычно представляемая в текстовой форме. Например, учебная или научная публикация, где новые понятия строятся на основании ранее определенных. В этом случае лингвистические переменные могут принимать новые, ранее не определенные значения, которые определяются контекстом — ближним (словосочетания) или общим (темой сообщения).

Возвращаясь к процедуре поиска как важнейшей составляющей использования баз данных, еще раз отметим, что критерий отбора должен содержать не только величину (например, слово), но и контекст.

В реальных системах поиск документальной информации[1], представленной в текстовой форме, производится по вторичным документам — специально создаваемым поисковым образам, точно идентифицирующим сам документ как единицу хранения и приблизительно, в краткой форме, путем *перечисления* основных понятий отражающим смысловое содержание. Такой подход позволяет построить процедуры поиска на основе теоретико-множественной модели с точной логикой отбора по критерию наличия заданного сочетания терминов запроса в списке терминов поискового образа. Однако контекст использования терминов должен быть доопределен отдельно — либо во время поиска, например указанием тематической области, либо после отбора из базы — во время ознакомления человека с содержанием найденного.

Определение контекста предметной области в целом осуществляется с помощью тезаурусов терминологических систем, фиксирующих с помощью родо-видовых и других отношений роль и семантику дескрипторов — выделенных терминов, которые используются для формирования поисковых образов документов.

Для доопределения смысла термина в составе поискового образа документа в первых поколениях автоматизированных информационных систем применялись специальные указатели роли, однако их ис-

[1] Это соответствует третьему из вышеперечисленных случаев. Два первых мы не рассматриваем, так как в этих случаях используются специализированные системы.

пользование было трудоемко и требовало специальной подготовки пользователя, поэтому в современных системах не применяется.

Другой важный фактор, влияющий на эффективность работы человека с информацией, — это форма хранения и представления — структура и оформление документа. Это особенно заметно при работе с объемными полнотекстовыми документами, причем иногда определяется на уровне машинного формата (например, DOC, PDF, HTML и т. д.), от выбора которого зависит возможность дальнейшей обработки.

В том случае, когда для хранения информации используются базы данных, структура документов может быть определена двумя путями[1]:

- так же как и для фактографических БД, заданием схемы — последовательности именованных типизированных полей данных;
- контекстным определением — использованием специализированных языков разметки (например, HTML или XML), задающих индивидуальные особенности представления материала каждого документа.

Использование встраиваемых определений структуры позволяет ввести «самоопределяемые» форматы представления документов. Это обеспечивает практически неограниченную гибкость при организации хранения коллекций разнородных документов, однако создает семантические проблемы согласованного использования материала (из-за возможности различной интерпретации определений), что в свою очередь требует создания доступного всем пользователям репозитария метаинформации — описаний природы и способов представления информации.

Контрольные вопросы и задания

1. Дайте определение понятиям «база данных» и «банк данных».
2. Каковы предпосылки создания баз и банков данных.
3. Перечислите преимущества и недостатки использования БнД.
4. Перечислите и определите назначение основных компонентов БнД.
5. Определите основные функции и назначение СУБД.

[1] Для реляционной СУБД MS SQL Server 2000 реализован импорт/ экспорт документов, представленных в XML-формате, в том числе с использованием схем сопоставления, определяющих отношение элементов XDR-схем к таблицам, а атрибутов — к столбцам.

6. Назовите отличительные особенности БнД.
7. Перечислите основные требования, предъявляемые к БнД.
8. Перечислите основные признаки классификации БД.
9. Определите понятие и назначение лингвистических средств БнД.
10. Перечислите основные категории пользователей БД.
11. Перечислите основные функции администратора БД.
12. Охарактеризуйте разницу между хорошо структурированной и слабо структурированной информацией.
13. Охарактеризуйте разницу между фактографическими и документальными БД.
14. Перечислите особенности OLAP- и OLTP-приложений.

Глава 2
ОСНОВЫ ФАКТОГРАФИЧЕСКИХ БД

Рассматриваемые в контексте понятия «информационная система» элементы реального мира, информацию о которых мы сохраняем и обрабатываем, будем называть *объектами*. Объект может быть материальным (например, служащий, изделие или населенный пункт) и нематериальным (например, имя, понятие, абстрактная идея). Будем называть *набором объектов* совокупность объектов, однородных с некоторой точки зрения (например, объектов *нашего* внимания, пусть даже и разнородных по своей внутренней природе).

Объект имеет различные свойства (например, цвет, вес, имя), которые важны для нас в то время, когда мы обращаемся к объекту (например, выбираем среди множества других) с какой-либо целью его использования. Причем свойства могут быть заданы как отдельными однозначно интерпретируемыми количественными показателями, так и словесными нечеткими описаниями, допускающими разную трактовку, иногда зависящую от точки зрения и наличных знаний воспринимающего субъекта.

Однако во всех случаях человек, работая с информацией, имеет дело с *абстракцией,* представляющей интересующий его фрагмент реального мира, — той совокупностью *характеристических свойств (атрибутов)*, которые важны для решения его прикладной задачи. Абстрагирование — это способ *упрощения* совокупности фактов, относящихся к реальному объекту (по своей сути бесконечно сложному и разнообразному при изучении его человеком). При этом некоторые свойства объекта игнорируются, поскольку считается, что для решения данной прикладной задачи (или совокупности задач) они не являются определяющими и не влияют на конечный результат действий при решении.

Цель такого абстрагирования — построение конструктивного операбельного описания (рабочей модели), удобного в обработке как

для человека, так и для машины, позволяющего организовать эффективную обработку больших объемов информации, причем высокопроизводительной должна быть работа не только вычислительной системы, но и взаимодействующего с ней человека.

2.1. Типология свойств и связей объекта

Природа характеристического свойства может быть различной. Рассмотрим основные типы свойств.

Свойство может быть *множественным* или *единичным* — т. е. атрибут, задающий свойство, может одновременно иметь несколько значений или соответственно только одно. Например, студент может владеть несколькими иностранными языками, но свойство «№ студенческого билета» будет иметь единственное значение.

Свойство может быть *простым* (не подлежащим дальнейшему делению с точки зрения прикладных задач) или *составным* — если его значение составляется из значений простых свойств. Например, свойство «Год рождения» является простым, а свойство «Адрес» — составным, так как включает значения простых свойств «Город», «Улица», «Дом», «Квартира».

В некоторых случаях полезно различать *базовые* и *производные* свойства. Например, объект СТУДЕНТ может иметь свойство «Общее количество сданных экзаменов», которое вычисляется суммированием сданных экзаменов.

Если наличие некоторого свойства для всех экземпляров сущности не является обязательным, то такое свойство называется *условным*. Например, не все сотрудники кафедры обладают свойством «Ученая степень» или «Ученое звание».

Значения свойств могут быть постоянными — *статическими* — или *динамическими*, т. е. меняться со временем. Например, свойство «№ студенческого билета» является статическим, а «Адрес» — может быть динамическим. Свойство может быть *неопределенным*, если оно является динамическим, но его текущее значение еще не задано.

Инструмент связей — это средство представления *сложных объектов*, каждый из которых может рассматриваться как множество некоторым образом взаимосвязанных *простых объектов*. Деление на простые и сложные объекты, так же как и характер взаимосвязи, является условным и определяется особенностями анализа предметной области, т. е. в конце концов — характером использования данных о пред-

метах в решаемых прикладных задачах. При этом с точки зрения, например, конструктора, ДЕТАЛЬ является сложным объектом, а с точки зрения поставщика — простым.

Среди многих разновидностей взаимосвязей наиболее частыми являются такие отношения иерархического типа, как «часть—целое», «род—вид».

Отношение «часть—целое» используется для представления *составных объектов*. Например, МАШИНА состоит из УЗЛОВ, УЗЕЛ состоит из ДЕТАЛЕЙ.

Отношение «род—вид» применяется для представления *обобщенных объектов*. Например, СОТРУДНИКИ подразделяются по профессии на КОНСТРУКТОРОВ, ПРОГРАММИСТОВ, РАБОЧИХ; ПРОГРАММИСТЫ — на ПРИКЛАДНЫХ ПРОГРАММИСТОВ и СИСТЕМНЫХ ПРОГРАММИСТОВ. Отношения «род—вид» обычно используются как основа классификации объектов по наборам характеристических признаков. Причем «видовые» объекты *наследуют* свойства «родовых».

Другой широко используемой разновидностью взаимосвязи является агрегирование — объединение простых объектов в сложный по принципу их принадлежности *агрегату* или их совместного участия в некотором процессе. Агрегирование, рассматриваемое здесь как более общий случай иерархических отношений, объединяет объекты разной природы с единственным общим свойством «совместное участие». Агрегированные объекты именуются обычно отглагольными существительными; например, «*Состав*»: ГРУППА *состоит из* СТУДЕНТОВ; «*Поставка*»: ПОСТАВЩИК *поставляет* ДЕТАЛИ.

2.2. Многоуровневые модели предметной области

Обычно отдельная база данных содержит (отражает) информацию о некоторой *предметной области* — наборе объектов, представляющих интерес для актуальных или предполагаемых пользователей. То есть реальный мир отображается совокупностью конкретных и абстрактных понятий, между которыми существуют (и соответственно фиксируются) определенные связи. Выбор для описания предметной области существенных понятий и связей является предпосылкой того, что пользователь будет иметь *практически все необходимые ему в рамках задачи* знания об объектах предметной области. Однако следует отметить, что пользователь, который хочет работать с базой данных, *дол-*

жен владеть основными понятиями, представляющими предметную область.

И в этом смысле абстрагирование позволяет построить такое описание (модель предметной области), которое другой человек сможет не только воспринять, но и безошибочно использовать для работы с описаниями экземпляров объектов, хранимых в базе данных. Модель предметной области соотносится с реальными объектами и связями так же, как схема маршрутов городского пассажирского транспорта с фактической траекторией движения автобуса. Схема адекватно отражает действительность на уровне основных понятий — маршрутов и остановок: выбрав по схеме маршрут, пассажир достигнет цели (прибудет на нужную остановку) независимо от того, в каком транспортном ряду будет двигаться автобус.

Наиболее простой способ представления предметных областей в БД реализуется поэтапно: 1) фиксацией *логической точки зрения* на данные (т. е. данные рассматриваются независимо от особенностей их хранения и поиска в конкретной вычислительной среде); 2) определением физического представления данных с учетом выбранных структур хранения данных и архитектуры ЭВМ.

Абстрагированное описание предметной области с фиксированной (логической) точки зрения будем называть *концептуальной схемой*. Само представление логической точки зрения, используемое при абстрагировании, — совокупность функциональных характеристик объектов и особенностей структурной организации данных, а также управления данными — будем называть *моделью данных*.

Отображение концептуальной схемы на физический уровень будем называть *внутренней схемой*.

Соотношение этих понятий приведено на рис. 2.1.

Отражение взгляда (точки зрения) отдельного пользователя на концептуальную схему (как вариант восприятия предметной области) будем называть *внешней схемой*. Внешняя схема использует те же абстрактные категории, что и концептуальная, а на практике соответствует логической организации данных в прикладной программе.

Теоретически вопрос о многообразии уровней абстракции был решен еще в 1960—1970-х гг. Основой для его решения является концепция многоуровневой архитектуры системы базы данных. Например, в отчете CODASYL [22] предусматривался архитектурный уровень подсхемы, который позволял для каждого конкретного приложения строить свое собственное «видение» используемого под-

множества базы данных путем определения его «персональной» подсхемы базы данных.

Рис. 2.1. Соотношение понятий концептуальной и внутренней схем

В более общем виде этот вопрос решен в архитектурной модели ANSI/X3/SPARC [20]. Здесь на внешнем уровне может поддерживаться совсем иная модель данных (или даже несколько моделей), чем на концептуальном уровне. Поддержка разнообразных возможностей абстрагирования в такой системе достигается благодаря средствам определения и поддержки межуровневого отображения моделей данных.

Помимо этого, для решения указанной проблемы может использоваться внутримодельная структура, например механизмы *представлений* (view). В объектных системах для этих целей может использоваться отношение наследования.

В общем случае концепция трехуровневого представления не требует более трех уровней, однако с практической точки зрения иногда удобно включать схемы дополнительных уровней. На рис. 2.2 приведены некоторые варианты решений.

На рис. 2.2, *б* выделена логическая схема, учитывающая особенности СУБД.

Пример, приведенный на рис. 2.2, *в*, характерен для варианта распределенной базы данных, объединяющей информацию, представленную разными внутренними схемами.

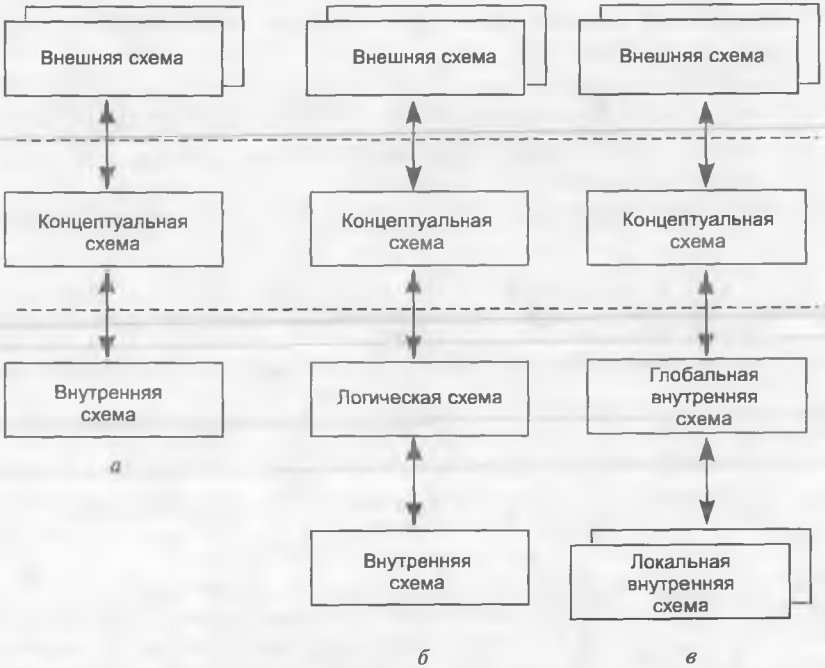


Рис. 2.2. Варианты решений трехуровневого представления

Рассмотренная трехуровневая архитектура обеспечивает выполнение основных требований, предъявляемых к системам баз данных:

- адекватность отображения предметной области;
- возможность взаимодействия с БД разных пользователей при решении разных прикладных задач;
- обеспечение независимости программ и данных;
- надежность функционирования БД и защита от несанкционированного доступа.

С точки зрения пользователей различных категорий трехуровневая архитектура имеет следующие достоинства:

- системный аналитик, создающий модель предметной области, не обязательно должен быть специалистом в области программирования и вычислительной техники;
- администратор баз данных, обеспечивающий отражение концептуальной схемы во внутреннюю, не должен беспокоиться о корректности представления предметной области;

- конечные пользователи, используя внешнюю схему, могут не вдаваться полностью в предметную область, обращаясь только к необходимым составляющим. При этом исключается возможность несанкционированного обращения к данным вне объявленных внешней схемой, так как формирование ее находится в сфере деятельности администратора базы данных;
- системный аналитик, как и конечный пользователь, не вмешивается во внутреннее представление данных.

Это отражает распространенную практику специализации и разделения ответственности. Главное же заключается в том, что работу по проектированию и эксплуатации баз данных можно разделить на три достаточно самостоятельных этапа. Хотя надо отметить, что на практике создание концептуальной схемы не всегда предшествует построению внешней. Иногда трудно с самого начала полностью определить предметную область, но, с другой стороны, уже известны требования пользователей (именно поэтому создание базы уже имеет смысл). И кроме того, адекватность модели предметной области в конце концов должна подтверждаться практикой пользовательских представлений.

2.3. Идентификация объектов и записей

В задачах обработки информации, и в первую очередь в алгоритмизации и программировании, атрибуты *именуют* (обозначают) и приписывают им *значения*.

При обработке информации мы так или иначе имеем дело с совокупностью объектов, *информацию о свойствах* каждого из которых надо сохранять (записывать) как *данные,* чтобы при решении задач их можно было найти и выполнить необходимые преобразования.

Таким образом, любое состояние объекта характеризуется совокупностью актуализированных атрибутов[1] (имеющих некоторое из значений в этот момент времени), которые фиксируются на некотором материальном носителе в виде *записи* — совокупности (группы) формализованных *элементов данных* (значений атрибутов, представленных в том или ином формате). Кроме того, в контексте задач хранения и поиска можно говорить, что значение атрибута *идентифици-*

[1] В общем случае объект может описываться совокупностью записей, относящихся к его составным частям или отражающих динамику изменения состояния.

рует объект: использование значения в качестве поискового признака позволяет реализовать простой критерий отбора по условию сравнения[1].

Так же как и в реальном мире, отдельный объект всегда уникален (уже хотя бы потому, что мы *именно его* выделяем среди других). Соответственно, запись, содержащая данные о нем, также должна быть узнаваема однозначно (по крайней мере, в рамках предметной области), т. е. иметь уникальный идентификатор, причем никакой другой объект не должен иметь такой же идентификатор. Поскольку идентификатор — суть значение элемента данных, в некоторых случаях для обеспечения уникальности требуется использовать более одного элемента. Например, для однозначной идентификации записей о дисциплинах учебного плана необходимо использовать элементы СЕМЕСТР и НАИМЕНОВАНИЕ ДИСЦИПЛИНЫ, так как возможно преподавание одной дисциплины в разных семестрах.

Предложенная выше схема представляет атрибутивный способ идентификации содержания объекта (рис. 2.3). Она является достаточно естественной для хорошо структурированных данных, имеющих фактографическую природу и описывающих обычно материальные объекты. Здесь важно отметить, что структурированность относится не только к форме представления данных (формат, способ хранения), но и к *способу интерпретации значения пользователем*: зна-

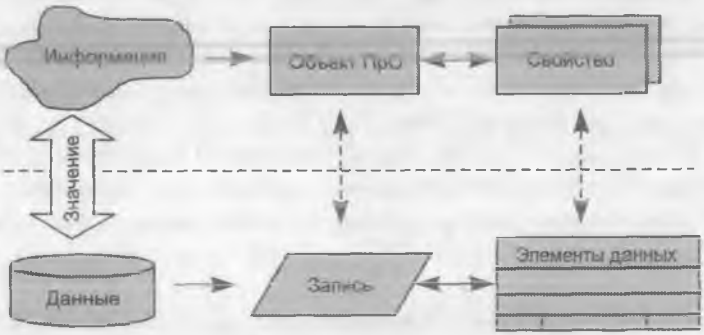


Рис. 2.3. Атрибутивный способ идентификации

[1] Следует отметить некоторые семантические проблемы идентификации через значение атрибута. Значение атрибута идентифицирует запись о **состоянии** объекта, и в случае изменения значения, например табельного номера служащего, будет невозможно ответить на вопрос: идет ли речь о том же служащем или о новом.

чение параметра не только представлено в предопределенной форме, но и обычно сопровождается указанием размерности величины, что позволяет пользователю понимать ее смысл без дополнительных комментариев. Таким образом, фактографические данные предполагают возможность их *непосредственной* интерпретации.

Однако атрибутивный способ практически не подходит для идентификации *слабо структурированных данных*, связанных с объектами, имеющими обычно *идеальную* (умозрительную) природу, — категориями, понятиями, знаковыми системами. Такие объекты зачастую определяются логически и опосредованно — через другие объекты. Для описания таких объектов используются естественные или искусственные языки, соответственно для понимания смысла пользователю необходимо использовать соответствующие правила языка, и, более того, часто необходимо уже располагать некоторой информацией, позволяющей идентифицировать и связать получаемую информацию с наличным знанием. То есть процесс интерпретации такого рода данных имеет *опосредованный* характер и требует использования дополнительной информации, причем такой, которая не обязательно присутствует в формализованном виде в базе данных.

Такое разделение нашло отражение в традиционном разделении баз данных на *фактографические* и *документальные*.

2.4. Поиск записей

Программисту или пользователю необходимо иметь возможность обращаться к отдельным, нужным ему записям (описаниям объектов) или отдельным элементам данных. В зависимости от уровня программного обеспечения прикладной программист может использовать следующие способы.

- Задать машинный адрес данных и в соответствии с физическим форматом записи прочитать значение. Это случай, когда программист должен быть «навигатором».
- Сообщить системе имя записи или элемента данных, которые он хочет получить, и, возможно, организацию набора данных. В этом случае система сама произведет выборку (по предыдущей схеме), но для этого она должна будет использовать вспомогательную информацию о структуре данных и организации набора. Такая информация по существу будет избыточной по отношению к объекту, однако общение с базой данных не будет

требовать от пользователя знаний программиста и позволит переложить заботы о размещении данных на систему.

В качестве ключа, обеспечивающего доступ к записи, можно использовать идентификатор — отдельный элемент данных. Ключ, который идентифицирует запись единственным образом, называется *первичным (главным)*.

В том случае, когда ключ идентифицирует некоторую группу записей, имеющих определенное общее свойство, ключ называется *вторичным* (*альтернативным*). Набор данных может иметь несколько вторичных ключей, необходимость введения которых определяется практической необходимостью — оптимизацией процессов нахождения записей по соответствующему ключу.

Иногда в качестве идентификатора используют составной *сцепленный ключ* — несколько элементов данных, которые в совокупности, например, обеспечат уникальность идентификации каждой записи набора данных или зададут общее свойство для группы записей.

При этом ключ может храниться в составе записи или отдельно. Например, ключ для записей, имеющих неуникальные значения атрибутов, для устранения избыточности целесообразно хранить отдельно. На рис. 2.4 приведены два таких способа хранения ключей и атрибутов для набора простейшей структуры.

Ключ	Атрибут
112	а/м ВАЗ 2110
113	а/м ВАЗ 2121
441	а/м ГАЗ 3110
456	а/м ВАЗ 2110
457	а/м ВАЗ 2121
678	а/м ВАЗ 2101
789	а/м ГАЗ 3110
889	а/м ВАЗ 2110
998	а/м ГАЗ 3102

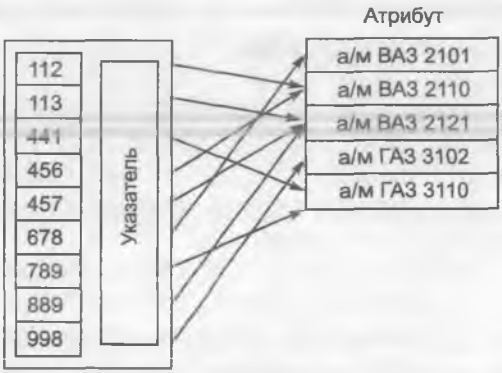


Рис. 2.4. Способы хранения ключа и атрибута

Введенное понятие ключа является логическим, и его не следует путать с физической реализацией ключа — *индексом*, обеспечивающим доступ к записям, соответствующим отдельным значениям ключа.

Один из способов использования вторичного ключа в качестве входа — организация инвертированного списка, каждый вход которого содержит значение ключа вместе со списком идентификаторов соответствующих записей. Данные в индексе располагаются в возрастающем или убывающем порядке, поэтому алгоритм нахождения нужного значения довольно прост и эффективен, а после нахождения значения запись локализуется по указателю физического расположения. Недостатком индекса является то, что он занимает дополнительное пространство и его надо обновлять каждый раз, когда удаляется, обновляется или добавляется запись. На рис. 2.5 приведен инвертированный список для предыдущего примера.

а/м ВАЗ 2110	678
а/м ВАЗ 2110	112, 456, 889
а/м ВАЗ 2121	113, 457
а/м ГАЗ 3102	998
а/м ГАЗ 3110	441, 789

Рис. 2.5. Инвертированный список для ключа «Марка автомобиля»

В общем случае инвертированный список может быть построен для любого ключа, в том числе составного.

В контексте задач поиска можно сказать, что существуют два основных способа организации данных. Первый соответствует примеру, приведенному на рис. 2.3, и представляет собой прямую организацию массива. Второй способ является инверсией первого, он соответствует рис. 2.4. Прямая организация массива удобна для поиска по условию «Каковы свойства указанного объекта?», а инвертированная — для поиска по условию «Какие объекты обладают указанным свойством?».

В [12] приводится следующая типология простых (атомарных) запросов:

1) **A(E) = ?** Каково значение атрибута A для объекта E?

2) **A(?) = V** Какие объекты имеют значение атрибута, равное V?

3) **?(E) = V** Какие атрибуты объекта E имеют значение, равное V?

4) **?(E) = ?** Какие значения атрибутов имеет объект E?

5) **A(?) = ?** Какие значения имеет атрибут A в наборе?

6) **?(?) = V** Какие атрибуты объектов набора имеют значение, равное V?

Здесь в запросах типов 2, 3, 6 вместо оператора равенства может быть использован другой оператор сравнения (*больше*, *меньше*, *не равно* или другие).

Запросы типа 1 выполняются поиском по «прямому» массиву: доступ к записи производится по первичному ключу. Запросы типа 2 выполняются поиском по инвертированному списку: доступ к записи(ям) производится по указателю, выбираемому из списка по значению вторичного ключа. Ответом в этих случаях будет *значение* атрибута или идентификатора. Запросы типа 3 имеют ответом *имя* атрибута.

Запросы типа 2, 5, 6 относятся к нескольким атрибутам, и в этом случае могут быть построены несколько индексов, облегчающих поиск по этим ключам.

Составные условия поиска могут использовать несколько простых условий, обычно связанных логическими (булевыми) операторами.

Следует отметить, что в контексте обработки запросов 2-го типа «Какие объекты имеют заданное значение атрибута?» можно выделить три следующих типа архитектур доступа.

1. *Системы с вторичными индексами.* В этих системах последовательность расположения записей соответствует последовательности значений первичного ключа. Как правило, используется один первичный индекс и несколько вторичных.

2. *Системы частично инвертированных файлов.* В этих системах записи могут располагаться в произвольной последовательности. В отличие от систем первого типа первичный индекс отсутствует. Вторичные индексы применяются для прямой адресации записей, что существенно облегчает включение в файл новых записей, так как допускается их размещение в любом свободном участке файла.

3. *Системы полностью инвертированных файлов.* В этих системах предусмотрено наличие файлов, содержащих значения отдельных элементов данных, входящих в состав записей, — допускается раздельное хранение элементов данных записи. Значения элементов данных, составляющих конкретную запись или кортеж, в общем случае могут размещаться в памяти произвольно. Для ускорения процесса поиска в системе используют два набора индексов: *индекс экземпляров* (значений ключей) и *индекс данных* (инвертированный список). С помощью индекса экземпляров можно найти в файле элементы данных, имеющих заданное значение. С помощью индекса данных можно найти записи, связанные с заданными значениями элементов. Такая организация характерна для организации данных *документальных информационных систем*.

2.5. Представление предметной области и модели данных

Если бы назначением базы данных было только хранение и поиск данных в массивах записей, то структура системы и самой базы была бы простой. Причина сложности в том, что практически любой объект характеризуется не только параметрами-величинами, но и взаимосвязями частей или состояний. Есть различия и в характере взаимосвязей между объектами предметной области: одни объекты могут использоваться только как характеристики остальных объектов, другие — независимы и имеют самостоятельное значение (рис. 2.6).

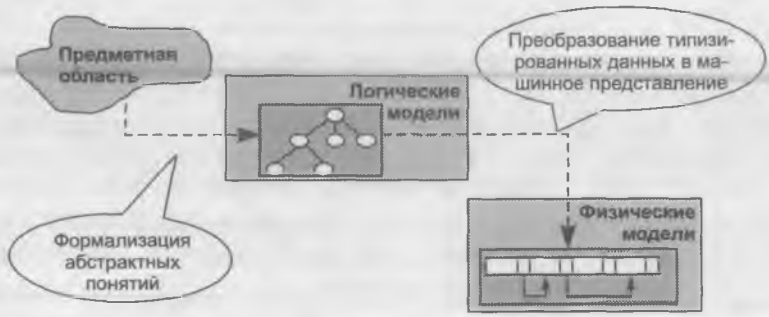


Рис. 2.6. Этапы преобразования представлений предметной области

Кроме того, сам по себе отдельный элемент данных (его значение) ничего не представляет. Он приобретает смысл только тогда, когда связан с атрибутом (природой значения, что позволит интерпретировать значение) и другими элементами данных.

Поэтому физическому размещению данных (и, соответственно, определению структуры физической записи) должно предшествовать описание логической структуры предметной области — построение *модели* соответствующего фрагмента реального мира, выделяющей только те объекты, которые будут интересны будущим пользователям, и представленные только теми параметрами, которые будут значимы при решении прикладных задач. Такая модель будет иметь очень мало физического сходства с реальностью, но будет полезна как *представление* пользователя о реальном мире. Причем это представление будет задаваться (описываться) *удобными для пользователя* средствами в не адекватной человеку жесткой вычислительной среде с двоичной логикой и числовым представлением информации.

Таким образом, прежде чем описывать физическую реализацию объектов и связей между ними, необходимо определить:

1) способ, с помощью которого внешние пользователи представляют (описывают) объекты и связи;

2) форму и методы внутримашинного представления элементов данных и взаимосвязей;

3) средства, обеспечивающие взаимно однозначные преобразования внешнего и внутримашинного представлений.

Такой подход является компромиссом, свойственным языкам программирования: за счет *предварительно определяемого множества абстракций*, общих для большинства задач обработки данных, обеспечивается возможность построения *надежных* программ обработки. Пользователь, используя *ограниченное множество формальных, но достаточно знакомых понятий*, выделяя сущности и связи, описывает объекты и связи предметной области; программист (или система автоматизации проектирования БД), используя такие *типовые абстрактные понятия* (как, например, числа, множества, последовательности, агрегаты), определяет соответствующие информационные структуры. Система управления данными, используя *двоичные формы представления типизированных* данных, обеспечивает эффективные процедуры хранения и обработки данных.

Именно введение *промежуточного* уровня абстракции позволяет иметь раздельное описание логического и физического представлений, освобождает конечного пользователя от необходимости беспокоиться о деталях внутримашинного представления и обработки, поскольку он может быть уверен, что программистом выбрана наиболее эффективная форма для данной ситуации. Однако эффективность здесь имеет определенные пределы. Чем ближе система абстракций к особенностям вычислительной среды, тем выше эффективность выполнения программы, но вынужденная «специализация» абстракций увеличивает вероятность того, что они станут неподходящими для некоторых других применений.

Модель данных должна так или иначе дать основу для описания данных и манипулирования данными, а также дать средства анализа и синтеза структур данных.

Необходимо отметить, что предметные среды с точки зрения описания целесообразно условно разделить на два полярных случая:

1. Предметная среда характеризуется сравнительно небольшим количеством типов отношений, но каждое отношение само есть большое

множество. Эти отношения сравнительно устойчивы, а изменений в пределах каждого множества существенно меньше мощности самого отношения. Например, отношение «вхождения» элементов изделий, содержащееся в конструкторских спецификациях, для среднего предприятия содержит сотни тысяч записей. В этом случае, задав схемы отношений и ориентировочные значения их мощностей, можно достаточно полно представить структуру и масштаб предметной среды.

2. Для предметной среды характерно большое число типов отношений между объектами, но каждое отношение есть множество сравнительно малой мощности. При этом мощность потока изменений для отношений сравнима с мощностью самих отношений.

Первый случай характерен для отображения процессов на уровне автоматизированных систем управления предприятиями. Современные системы управления базами данных наиболее эффективны именно в подобном случае, при отображении статических в указанном смысле предметных сред. Обычно при этом речь идет о целых классах объектов, например о деталях данного типа, и не отображается состояние каждой конкретной детали.

Второй случай характерен для описания производственного *технологического процесса* с учетом временных и пространственных факторов нахождения конкретных объектов.

Если в первом случае говорят о реляционной, иерархической или сетевой моделях данных, то во втором — о семантических сетях и фреймах.

Основное отличие этих методов заключается в том, что первые задают четкую схему (так называемую схему базы данных), в рамках которой и отображается предметная область. Подобное построение по сути своей является довольно статичным, требует априорного знания типов отношений, в которых может находиться объект, однако зафиксированная схема базы данных позволяет довольно эффективно организовать поиск необходимой информации. Во втором случае предметная среда отображается (по крайней мере, на уровне модели) в виде однородной сети, любые изменения которой по вводу как новых классов объектов, так и новых типов отношений не связаны с какими-либо структурными преобразованиями сети. В силу большого количества типов отношений манипулирование подобной «элементарной» информацией достаточно затруднено, поэтому для данного случая характерно введение большого количества общих понятий (и соответствующих им отношений), что упрощает работу с таким представлением.

В контексте машинного представления модель данных может быть использована следующим образом:

- как средство спецификации типов данных и их организации, разрешенных в конкретной БД;

- как основа разработки общей методологии построения баз данных;
- как основа минимизации влияния эволюции баз данных на уже существующие прикладные программы и работу конечных пользователей;
- как основа разработки семейства языков запросов и языков манипулирования данными;
- как основа архитектуры СУБД;
- как основа изучения динамических свойств различных организаций данных.

Таким образом, модель данных — это базовый инструментарий, обеспечивающий на формальном абстрактном уровне конкретные способы представления объектов и связей.

Модель базы данных охватывает более широкий спектр понятий. Основное назначение модели базы данных состоит в том, чтобы:

- определить ясную границу между логическим и физическим аспектами управления базой данных (*независимость данных*);
- обеспечить конечным пользователям и программистам, создающим БД, возможность и средства общего понимания смысла данных (*коммуникабельность*);
- определить языковые понятия высокого уровня, обеспечивающие возможность выполнения однотипных операций над большими совокупностями записей (в общем случае разнотипных данных) как единую операцию (*обработка множеств*).

2.6. Основные понятия реляционной модели данных

Реляционная[1] модель является удобной и наиболее привычной формой представления данных в виде таблицы. В отличие от иерархической и сетевой модели, такой способ представления: 1) понятен пользователю-непрограммисту; 2) позволяет легко изменять схему — присоединять новые элементы данных и записи без изменения соответствующих подсхем; 3) обеспечивает необходимую гибкость при обработке непредвиденных запросов. К тому же любая сетевая или

[1] В математических дисциплинах понятию «таблица» соответствует понятие «отношение» (relation). Отсюда и произошло название модели — реляционная. То есть применительно к базам данных понятия «реляционная БД» и «табличная БД» являются по существу синонимами.

иерархическая схема может быть представлена двумерными отношениями.

Функциональным назначением реляционной модели является обеспечение согласованности структур данных, операций манипулирования данными, целостность данных.

Одним из основных преимуществ реляционной модели является ее однородность. Все данные рассматриваются как хранимые в таблицах, в которых каждая строка имеет один и тот же формат. Каждая строка в таблице представляет некоторый объект реального мира или соотношение между объектами. Пользователь модели сам должен для себя решить вопрос, обладают ли соответствующие сущности реального мира однородностью. Этим самым решается проблема пригодности модели для предполагаемого применения.

Основными понятиями, с помощью которых определяется реляционная модель, являются следующие: *домен, отношение, кортеж, кардинальность, атрибут, степень отношения, первичный ключ*. Соотношение этих понятий иллюстрируется рис. 2.7. Эти понятия представляют специальную терминологию, введенную авторами теоретических основ, однако они имеют и более привычные аналоги (но не во всем эквиваленты!), соответствие которых приведено в табл. 2.1.

Домен — это множество возможных значений, из которого берутся значения соответствующего атрибута определенного отношения. С точки зрения программирования домен — это тип данных, определяемый системой (стандартный) или пользователем.

Рис. 2.7. Основные понятия реляционной модели

Таблица 2.1. **Соотношение понятий**

Отношение	Таблица
Домен	Совокупность допустимых значений
Кортеж	Строка таблицы
Кардинальность	Количество строк в таблице
Атрибут	Поле, столбец таблицы
Степень отношения	Количество полей (столбцов)
Первичный ключ	Уникальный идентификатор

Отношение имеет две части — заголовок и тело. Заголовок — это множество поименованных атрибутов, каждый из которых задан на определенном домене, а тело — это множество кортежей, содержащих значения атрибутов.

Отношения (таблицы) реляционной модели удовлетворяют следующим требованиям:

- каждый атрибут ассоциирован с определенным доменом (типом данных);
- каждый атрибут уникально поименован и содержит текущее значение этого атрибута;
- каждое значение любого атрибута является атомарным;
- отсутствуют одинаковые строки.

Атрибуты и кортежи отношения размещаются в произвольном порядке.

Первичный ключ — это атрибут или некоторое подмножество атрибутов, которые уникально, т. е. единственным образом, идентифицируют кортежи внутри отношения. Первичный ключ, который состоит более чем из одного атрибута, называется составным (множественным, комбинированным). Правило целостности объектов утверждает, что первичный ключ не может быть полностью или частично пустым, т. е. иметь значение null.

Остальные ключи, которые можно также использовать в качестве первичных, называются *потенциальными* или *альтернативными* ключами.

Внешний ключ — это атрибут (или некоторое подмножество атрибутов) одного отношения, который может служить в качестве первичного ключа для другого отношения. Внешний ключ тем самым является ссылкой, обеспечивающей связь кортежей двух отношений с использованием первичного ключа. Очевидно, что внешний ключ, как

и первичный, может представлять собой комбинацию атрибутов, и количество столбцов и их типы данных в первичном и внешнем ключах совпадают.

2.7. Основы реляционной алгебры

С точки зрения внешнего представления (абстрагирования на логическом уровне) объектов реального мира *модель данных* — это основные понятия и способы, используемые при анализе и описании предметной области.

Среди многих попыток представить обработку данных на формальном абстрактном уровне реляционная модель, предложенная Э.Ф. Коддом, стала по существу первой *работоспособной* моделью данных, поскольку помимо средств описания объектов имела эффективный инструментарий преобразований этих описаний — операции реляционной алгебры.

Реляционная алгебра может быть задана множеством T отношений (реляционных таблиц) и множеством O алгебраических операций над ними. Все операции из множества O переводят операнды-отношения из T в отношения, принадлежащие T.

К числу операций реляционной алгебры (в том виде, в котором она была определена Э.Ф. Коддом) относят следующие:

- бинарные операции:
  $\cup$ — объединение;
  $\cap$ — пересечение;
  $\setminus$ — разность;
  $\times$ — декартово произведение;
  J — соединение;
  / — деление;
- унарные операции:
  S — выборка;
  Pr — проекция.

Этот набор операций можно классифицировать так:

- теоретико-множественные операции (аналогичные одноименным операциям в теории множеств, но модифицированные с учетом того, что их операндами являются отношения): объединение, пересечение, разность, декартово произведение;
- специальные реляционные операции: выборка, проекция, соединение, деление.

Рассмотрим подробнее операции реляционной алгебры.

Объединение (∪) возвращает отношение, содержащее все кортежи, которые принадлежат либо одному из двух заданных отношений, либо им обоим (рис. 2.8).

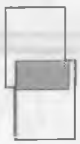
Пересечение (∩) возвращает отношение, содержащее все кортежи, которые принадлежат одновременно двум заданным отношениям (рис. 2.9).

FIO	Year	Job	Chair
Цветкова С.С.	1960	Доцент	23
Козлов К.К.	1959	Доцент	23
Петров П.П.	1960	Ст. преп.	24
Лютикова Л.Л.	1977	Ассистент	24

FIO	Year	Job	Chair
Иванов И.И.	1948	Зав. каф.	22
Сидоров С.С.	1953	Проф.	22
Гиацинтова Г.Г	1945	Проф.	22
Цветкова С.С.	1960	Доцент	23
Козлов К.К.	1959	Доцент	23

FIO	Year	Job	Chair
Иванов И.И.	1948	Зав. каф.	22
Сидоров С.С.	1953	Проф.	22
Гиацинтова Г.Г	1945	Проф.	22
Цветкова С.С.	1960	Доцент	23
Козлов К.К.	1959	Доцент	23
Петров П.П.	1960	Ст. преп.	24
Лютикова Л.Л.	1977	Ассистент	24

Рис. 2.8. Объединение



FIO	Year	Job	Chair
Иванов И.И.	1948	Зав. каф.	22
Сидоров С.С.	1953	Проф.	22
Гиацинтова Г.Г	1945	Проф.	22
Цветкова С.С.	1960	Доцент	23
Козлов К.К.	1959	Доцент	23

FIO	Year	Job	Chair
Цветкова С.С.	1960	Доцент	23
Козлов К.К.	1959	Доцент	23

FIO	Year	Job	Chair
Цветкова С.С.	1960	Доцент	23
Козлов К.К.	1959	Доцент	23
Петров П.П.	1960	Ст. преп.	24
Лютикова Л.Л.	1977	Ассистент	24

Рис. 2.9. Пересечение

Разность (\) возвращает отношение, содержащее все кортежи, которые принадлежат первому из двух заданных отношений и не принадлежат второму (рис. 2.10).

Декартово произведение (×) возвращает отношение, содержащее все возможные кортежи, которые являются сочетанием двух кортежей, принадлежащих соответственно двум заданным отношениям (рис. 2.11).

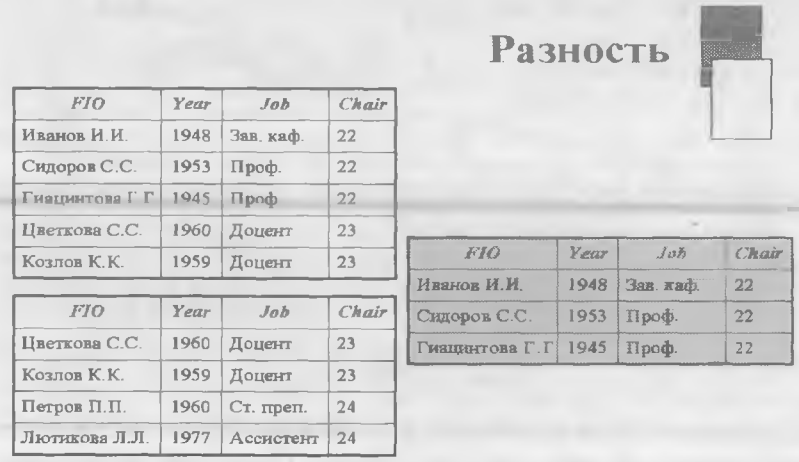


Рис. 2.10. Разность

Рис. 2.11. Декартово произведение

Выборка. Пусть задано логическое выражение — условие отбора F с атрибутами некоторого реляционного отношения R. Тогда результатом выборки $S_F(R)$ будет подмножество множества записей R, для которых F имеет значение *истина* (рис. 2.12).

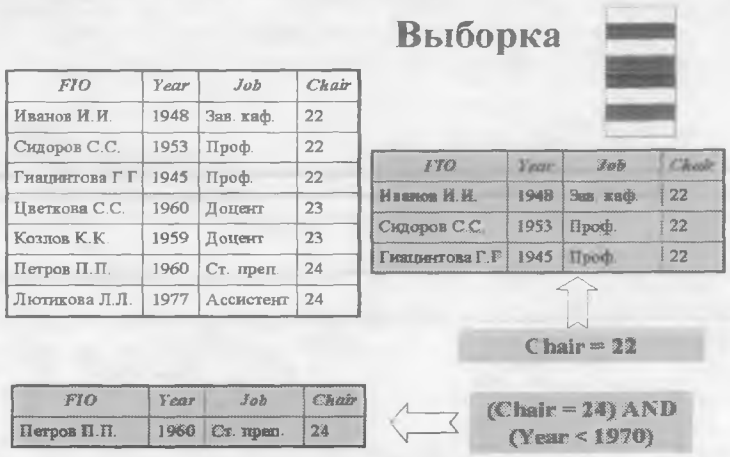


Рис. 2.12. Выборка

Проекция. Пусть отношение R имеет n атрибутов. Зададим подмножество атрибутов $a_1, a_2, ..., a_m$ ($m \le n$). Тогда результатом проекции $Pr_{a_1, a_2, ..., a_m}(R)$ будет реляционное отношение, имеющее m атрибутов из n атрибутов отношения R. Следует отметить, что в результате операции количество кортежей может уменьшиться, так как по определению отношение не может содержать одинаковые кортежи (рис. 2.13).

Соединение. Рассмотрим вариант так называемого естественного соединения двух реляционных таблиц по равенству значений заданных атрибутов. Будем считать, что в отношениях сравниваются одноименные атрибуты. Тогда естественным соединением отношений R и P по равенству атрибутов $a_1, a_2, ..., a_m$ (каждый из которых присутствует в обоих отношениях) будет отношение, полученное после выборки из декартова произведения отношений R и P только тех кортежей, на которых значения заданных в операции атрибутов совпадают. При этом значения одноименных атрибутов в результирующем кортеже появляются один раз, а не дважды (рис. 2.14).

Деление. Отношение, полученное в результате деления R/P, содержит в качестве атрибутов те и только те атрибуты делимого R, ко-

Проекция

FIO	*Year*	*Job*	*Chair*
Иванов И.И.	1948	Зав. каф.	22
Сидоров С.С.	1953	Проф.	22
Гиацинтова Г.Г	1945	Проф.	22
Цветкова С.С.	1960	Доцент	23
Козлов К.К.	1959	Доцент	23
Петров П.П.	1960	Ст. преп.	24
Лютикова Л.Л.	1977	Ассистент	24

FIO	*Job*
Иванов И.И.	Зав. каф.
Сидоров С.С.	Проф.
Гиацинтова Г.Г	Проф.
Цветкова С.С.	Доцент
Козлов К.К.	Доцент
Петров П.П.	Ст. преп.
Лютикова Л.Л.	Ассистент

Рис. 2.13. Проекция

Соединение

FIO	*Job*	*Chair*
Иванов И.И.	Зав. каф.	22
Сидоров С.С.	Проф.	22
Гиацинтова Г.Г	Проф.	22
Цветкова С.С.	Доцент	23
Козлов К.К.	Доцент	23
Петров П.П.	Ст. преп.	24
Лютикова Л.Л.	Ассистент	24

Job	*Pay*
Зав. каф.	3000
Проф.	2500
Доцент	2000
Ст. преп.	1500
Ассистент	1200

FIO	*Job*	*Chair*	*Pay*
Иванов И.И.	Зав. каф.	22	3000
Сидоров С.С.	Проф.	22	2500
Гиацинтова Г.Г	Проф.	22	2500
Цветкова С.С.	Доцент	23	2000
Козлов К.К.	Доцент	23	2000
Петров П.П.	Ст. преп.	24	1500
Лютикова Л.Л.	Ассистент	24	1200

Рис. 2.14. Соединение

торые отсутствуют в делителе P, а в качестве кортежей в результат деления включаются те кортежи делителя, которые при декартовом умножении частного на делитель P содержатся в делимом R (рис. 2.15).

Результат выполнения любой операции над отношением также является отношением, поэтому результат одной операции может ис-

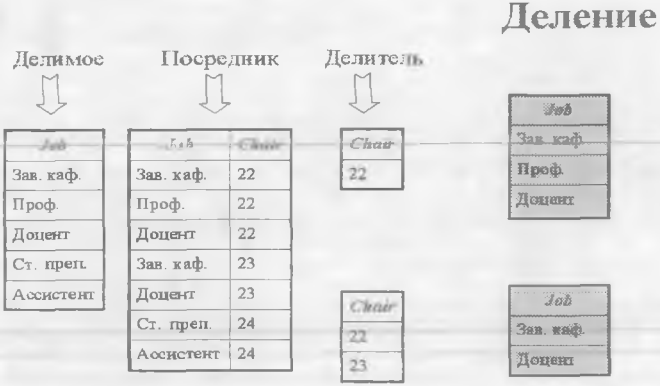


Рис. 2.15. Деление

пользоваться в качестве исходных данных для другой. Другими словами, можно записывать вложенные реляционные выражения, т. е. выражения, в которых операторы сами представлены реляционными выражениями, причем произвольной сложности. Эта особенность называется свойством *реляционной замкнутости*.

Реляционная алгебра имеет набор правил вывода типов (отношений), позволяющих вывести тип (отношение) на выходе произвольной реляционной операции, зная типы (отношения) на входе этой операции. Задав такие правила для всех операций, можно гарантировать, что для реляционного выражения любой сложности будет вычисляться результат, имеющий вполне определенный тип (отношение) и, в частности, известный набор имен атрибутов.

Рассмотренные восемь операторов Кодда не являются минимальным набором, так как не все из них примитивны, т. е. часть из них можно определить через другие операторы. Действительно, операции соединения, пересечения и деления можно определить через остальные пять. Эти пять операций (выборка, проекция, произведение, объединение и разность) можно рассматривать как примитивные в том смысле, что ни одна из них не выражается через другие. Они образуют минимальный набор, но, тем не менее, необязательно единственно возможный. Кроме того, остальные три операции (в особенности операция соединения) на практике используются настолько часто, что, несмотря на то что они не являются примитивными, имеет смысл обеспечить их непосредственную поддержку.

Предшествующее рассмотрение алгебры представлено в контексте только операций выборки данных. Однако, как отмечается в клас-

сических введениях к описаниям реляционной алгебры, ее основная цель — обеспечить запись реляционных выражений, позволяющих определять:

- области выборки, т. е. тех данных, которые должны быть доставлены в результате выполнения операции выборки;
- области обновления, т. е. данных, которые должны быть вставлены, изменены или удалены в результате выполнения операции обновления;
- правила поддержки целостности данных, т. е. некоторых особых требований, которым должна удовлетворять база данных;
- производные переменные-отношения, т. е. те данные, которые должны быть включены в представления базы данных;
- требования устойчивости, т. е. данные, которые должны быть включены в контролируемую область для некоторых операций управления параллельным доступом к информации;
- ограничения защиты, т. е. данные, для которых осуществляется тот или иной тип контроля доступа.

В целом выражения реляционной алгебры служат для символического высокоуровневого представления намерений пользователя (например, в отношении некоторого определенного запроса). И именно потому, что подобные выражения являются символическими и высокоуровневыми, ими можно манипулировать в соответствии с различными правилами высокоуровневых преобразований, в том числе и для оптимизации процедур выполнения запросов на данные.

2.8. Реляционное исчисление

Реляционная алгебра — это процедурный язык высокого уровня, который может применяться в СУБД для последовательного *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. С формальной точки зрения реляционная алгебра и реляционное исчисление эквивалентны, т. е. для каждого выражения алгебры имеется эквивалентное выражение исчисления (и наоборот).

В выражениях реляционной алгебры всегда явно задается некий порядок и подразумевается некая стратегия вычисления запроса (т. е. *как* необходимо извлечь данные). В реляционном исчислении же

не существует описания процедуры вычисления запроса, поскольку в запросе реляционного исчисления указывается, *что* следует извлечь.

Название «Реляционное исчисление» произошло от *исчисления предикатов*. В контексте баз данных оно существует в двух формах: в форме *реляционного исчисления кортежей*, предложенного Коддом, и в форме *реляционного исчисления доменов* (Лакруа и Пиро).

Под предикатом в теории исчисления предикатов подразумевается истинностная функция с параметрами. После подстановки значений вместо параметров функция становится выражением, называемым *суждением,* которое может быть истинным или ложным. Например, предложение «Иванов И.И. является сотрудником данного вуза» является суждением, поскольку можно определить его истинность или ложность.

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.

Предикаты могут соединяться с помощью логических операций AND, OR и NOT с образованием составных предикатов.

В реляционном исчислении кортежей задача состоит в нахождении таких кортежей, для которых предикат является истинным. Это исчисление основано на переменных кортежа. Переменные кортежа — это переменные, областью определения которых является указанное отношение, т. е. допустимыми значениями для них 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).

В реляционном исчислении доменов также используются переменные, но их значения берутся из области определения атрибутов, а не из кортежей отношения.

Контрольные вопросы

1. Дайте определение понятия предметной области.
2. Что является результатом абстрагированного описания предметной области?
3. Приведите варианты модели трехуровнего представления ПрО.

4. Дайте определение атрибутивного способа идентификации объектов и записей.
5. Приведите типологию простых запросов.
6. Определите понятия первичного и вторичного ключа записи.
7. Определите основные требования к *модели данных*
8. Дайте сравнительную характеристику понятий *модель данных* и *концептуальная схема*.
9. Перечислите операции реляционной алгебры.
10. Дайте определение реляционных операций соединения, пересечения и деления через пять других операций.
11. Докажите ассоциативность и коммутативность реляционной операции объединения.
12. Являются ли реляционные операции умножения и деления взаимообратными?
13. Перечислите основные свойства реляционной структуры данных.
14. Охарактеризуйте разницу между реляционной алгеброй и реляционным исчислением.

Глава 3
БАЗОВЫЕ ТЕХНОЛОГИИ И ОСНОВНЫЕ ЭТАПЫ РАЗВИТИЯ МАШИННОЙ ОБРАБОТКИ ДАННЫХ

3.1. Введение в технологии машинной обработки данных и основные определения

Реальные базы данных промышленного масштаба содержат миллионы записей, данные которых описывают состояния и взаимосвязи многих и многих объектов реального мира. Требования, предъявляемые пользователями к автоматизированным или автоматическим системам, обрабатывающим эти данные, обусловливают и требования к параметрам внешней памяти, в первую очередь к высокой оперативности доступа.

Важной особенностью здесь является то, что архитектура систем и технологий управления данными непосредственно связана с двумя следующими значительными, хотя и противоположными обстоятельствами:

- непредсказуемой вариантностью представления данных в прикладной программе, зависящей от разнообразных особенностей пользовательских задач;
- жесткостью технических решений устройств внешней памяти, выражающейся в функциональной простоте[1] операций и ограниченности форм представления данных.

[1] Требование операционной простоты определяется производственными и экономическими причинами: устройство должно быть надежным в использовании и дешевым в изготовлении (т. е. содержать минимум механических компонент и сложной логики).

Высокая эффективность решений в области обработки данных достигается введением промежуточных слоев специализированных технических и программных средств. Характер проблем и архитектурно-технологические решения такого рода достаточно полно иллюстрируются приведенной на рис. 3.1 примерной схемой реализации операций ввода-вывода — взаимодействия прикладной программы с компонентами операционной системы и устройствами внешней памяти. Здесь *специализация* компонентов выражается в том, что по существу каждый из них реализует различные способы работы с потоком данных (и в частности, его фрагментацию на блоки), что и обеспечивает, с одной стороны, необходимый уровень декомпозиции и идентификации логических/физических записей, а с другой — независимость физического и логического уровней представления данных.

Здесь термины *логический* и *физический* отражают различия аспектов представления данных. *Логическое представление* указывает на то, как данные используются в прикладной программе, т. е. отражают логику обработки. *Физическое представление* — это то, как данные хранятся на *физическом носителе*.

Будем считать *логической записью* идентифицируемую (именованную) *совокупность элементов или агрегатов данных*, воспринимаемую прикладной программой как единое целое при обмене информацией с внешней памятью (по крайней мере, для операций ввода-вывода).

Физической записью будем считать совокупность данных, которая может быть считана или записана как единое целое *одной командой ввода-вывода*. Важно, что для компонентов различного уровня в технологической цепи ввода-вывода состав и структура физической записи может быть разной.

Структура данных и их взаимосвязь в случаях логического и физического представления могут не совпадать. Например,

а) одна физическая запись может включать несколько логических;

б) порядок следования элементов данных в физической записи может быть изменен для оптимизации использования пространства памяти.

То есть если логическая структура может варьироваться в широком диапазоне и даже представляться, например, вариантными записями, то физическая — практически всегда представлена жесткой структурой, причем в значительной степени определяемой типом носителя.

Рис. 3.1. Примерная схема организации ввода-вывода

3.2. Примерная схема организации файлового ввода-вывода

Рассмотрим для представленной на рис. 3.1 схемы ввода-вывода способы адресации и последовательность операций[1] выборки данных, обеспечивающих чтение прикладной программой с тома внешней памяти (например, магнитного диска ПЭВМ) некоторой произвольной (*i*-й) записи. Отметим еще раз, что «специализация» компонентов, участвующих в операциях ввода-вывода, выражается прежде всего в используемом способе адресации.

Прикладная программа использует одномерную (или сводимую к одномерной) сквозную адресацию данных на уровне логических записей: запись определяется номером, например соответствующим порядку ее размещения.

Система управления физическим вводом-выводом (в рассматриваемом примере — BIOS ПЭВМ) использует трехмерную систему координат: адрес записи составляется из номера дорожки, номера головки чтения-записи (номер поверхности) и номера сектора. Операционная система же использует одномерную *сквозную* систему координат: сектора нумеруются от края диска к центру последовательно, причем сначала в рамках одного сегмента цилиндра (кластера), далее сектора следующего сегмента дорожки, после чего происходит переход к следующей дорожке.

Этот способ адресации и, соответственно, порядок использования пространства отчасти отражает специфику аппаратных решений, ориентированных на временную оптимизацию операций ввода-вывода: большее количество данных будет считано при одном обращении к диску за счет одновременного обращения через головки чтения-записи к данным, размещенным на параллельных дорожках в одном секторе одного цилиндра. Фиксированное количество битов, равное размеру сектора, определенного при разметке, умноженному на число головок, будет прямо (без дополнительной обработки, например проверки логических условий конца файла или записи) передано в буфер оперативной памяти устройства или операционной системы.

---

[1] Здесь не рассматриваются такие технологии ввода-вывода, как поточное или чтение с опережением, отображение файла в память и т. п., которые по существу являются оптимизационными, выполняемыми специальными средствами ОС.

Также, в целях общности, в этом примере не рассматриваются подготовительные операции, такие как открытие файла и выделение памяти для рабочих и системных буферов, хотя они также достаточно ресурсоемки.

Таким образом, если система адресации в прикладной программе является относительной и отражает логику взаимосвязи записей (например, порядок создания файла), то для подсистем ввода-вывода она является абсолютной и определяется *физическим форматом носителя*: размером сектора, количеством секторов на дорожке, количеством поверхностей и дорожек и т. д. При этом независимость от особенностей физического размещения и механизма адресации обеспечивается на уровне *логической структуры носителя*.

Например, логически последовательная выборка записей файла обеспечивается таблицей размещения файлов, определяющей используемое файлом пространство как цепочку кластеров, физически находящихся в любой доступной части диска. Доступ к файлу производится по идентификатору (составному имени) через систему каталогов, связывающих идентификатор файла с началом цепочки указателей на кластеры данных в таблице размещения файлов. Кроме того, логическая структура содержит (в составе загрузочной записи) информацию, идентифицирующую пространство в целом, а также данные, *определяющие физическую структуру* (физический формат носителя, рассмотренный ранее).

В общем случае операция чтения физической записи включает следующие действия.

1. Определение адреса записи в координатах устройства (например, для файлов с записями фиксированной длины — пересчет номера нужной записи в относительный адрес сектора и далее определение абсолютного номера сектора на диске).

2. Перемещение головки чтения в соответствующую координату: позиционирование дорожки и сектора на дорожке, складывающееся из двух действий — собственно радиального перемещения головки на расстояние от текущего положения до нужной дорожки и ожидания подхода указанного сектора вращающегося диска к позиции, где находится головка. Следует также отметить, что высокая плотность записи данных означает, что промежуток между секторами[1] и дорожками сравнительно мал (сопоставим с погрешностями механизма переме-

[1] Если контроллер не успевает завершить обработку передачи и подготовиться к передаче данных, размещаемых на физически следующем секторе, то придется ожидать завершения полного оборота диска. С целью исключения таких потерь диск форматируется так, что логически последовательные секторы разделены одним или несколькими физическими секторами (коэффициент чередования), тогда контроллер будет готов выполнить операцию со следующим логическим сектором, не ожидая дополнительного оборота.

щения и тепловым расширением), и поэтому правильность позиционирования определяется по служебным данным заголовка[1] сектора, считываемым до начала передачи прикладных данных.

3. Пересылка данных, расположенных в области кластера, в буфер, который физически может быть как частью устройства, так и областью оперативной памяти.

4. Завершение операции (проверка корректности чтения, например по контрольной сумме) и возврат управления ОС для обработки считанных данных.

5. Выделение системой данных, относящихся к затребованным записям. Причем во многих случаях в системный буфер считываются не только данные логической записи, нужные прикладной программе, но и соседние. Это позволяет сократить суммарные затраты времени при чтении нескольких записей, исключив наиболее долгую операцию позиционирования. Указание на такое *блокирование* может выдаваться явно прикладной программой при открытии файла или операционной системой, использующей собственные механизмы кэширования для оптимизации[2] ввода-вывода.

6. Передача в рабочую область прикладной программы данных запрошенной ею логической записи или указателя на соответствующую область памяти в системном буфере.

В этой последовательности наиболее медленными операциями являются механическое позиционирование головок и чтение данных с поверхности носителя (эти операции выполняются на порядки медленнее, чем операции пересылки). Поэтому выигрыш во времени может быть получен только в случае выполнения серии запросов на доступ к данным, причем экономия может достигаться следующими путями.

1. Суммарным сокращением перемещения головок за счет организации такой последовательности обращения к записям (или такого

---

[1] Такой подход форматирования (разметки) пространства внешней памяти используется и в случае таких устройств «истинно» последовательного доступа, как магнитные ленты, для обеспечения ускоренного «прямого» доступа к сектору по его номеру — прямому адресу (еще с тех времен, когда не были созданы дисковые накопители, например ЭВМ второго поколения «Минск-22»). При этом, поскольку данные секторов, предшествующих нужному, передавать не надо, позиционирование будет выполняться с максимальной скоростью перемещения ленты в режиме «перемотки».

[2] Автоматическое использование системы кэширования и упреждающего чтения (не учитывающее особенности порядка обращения к данным, обусловленного алгоритмом обработки) может привести к обратному результату, например в случае обращения к логическим записям в произвольной последовательности (случайной), не соответствующей физическому следованию записей.

порядка их физического размещения), когда перемещение от текущего положения к следующему будет минимальным.

2. Формированием логических записей таким образом, чтобы их формат (длина данных) соответствовал физическому формату хранения. В случае кратности длин, т. е. если длина логической записи будет кратной длине кластера или в кластере будет размещаться целое число записей, будет исключена передача данных, не запрошенных текущей операцией.

Непосредственное применение приведенных методов повышения эффективности, тем не менее, достаточно ограничено по целому ряду причин. По мере добавления новых типов данных или при появлении новых приложений структура записей должна будет меняться — возникает необходимость переписывать и отлаживать прикладные программы. Чем больше количество прикладных программ имеется в наличии, тем более дорогой становится эта процедура.

Основой для практического решения этой проблемы является принцип выделения и представления описательных составляющих в виде самостоятельных операционных объектов, хранимых отдельно от определяемых ими данных.

3.3. Эволюция концепций обработки данных

Характер возможных представлений данных и архитектурные решения, отражающие степень специализации компонентов управления, хорошо иллюстрируются представленной в [12] эволюцией концепций обработки данных.

С появлением в конце 1960-х гг. понятия *база данных* взаимосвязь файлов (логических) и наборов данных (физических файлов) рассматривается в контексте *неизбыточности и независимости данных, их защиты и возможности доступа в реальном времени.*

3.3.1. Простые (линейные) файлы данных (начало 1960-х гг.)

Для линейных «простых» файлов организация хранения и доступа характеризуется следующими особенностями (рис. 3.2):

- записи в файлах размещаются и обрабатываются последовательно. Физическая структура хранения данных точно такая же, как логическая;

Рис. 3.2. Линейные файлы данных

- программное обеспечение ввода-вывода выполняет только операции физического чтения-записи. При обновлении отдельной записи файл всегда перезаписывается на другой носитель, а предыдущие поколения данных сразу не уничтожаются;
- прикладной программист определяет физическое расположение данных и включает формирование физической структуры в прикладные программы. Если структура данных или запоминающее устройство изменяется, прикладную программу необходимо переписать;
- наборы данных обычно создаются и оптимизируются для одного приложения. Одни и те же данные редко используются для нескольких приложений.

3.3.2. Методы доступа к записям (конец 1960-х гг.)

Этот этап характеризуется изменением природы файлов и устройств. Появляются дисковые устройства с прямым доступом и возможностью обновления «по месту изменений», а программное обес-

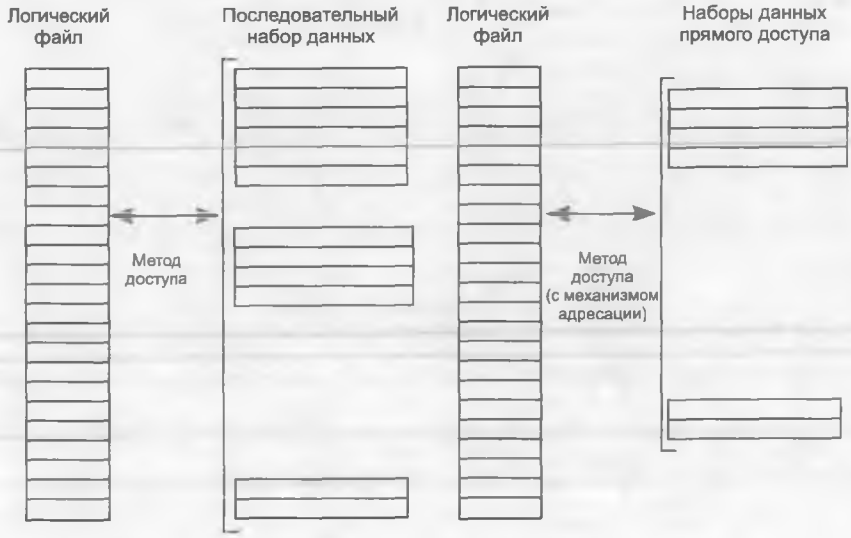


Рис. 3.3. Методы доступа к записям

печение позволяет без перекомпиляции программы изменять расположение набора данных, но без изменения структуры записей и типа организации набора (рис. 3.3).

Организация хранения и доступа в этом случае характеризуется следующими особенностями:

- логическая и физическая структуры файла различаются между собой, но взаимосвязь между ними достаточно простая. Запоминающее устройство можно менять без изменения прикладной программы;
- файл создается в прикладной программе как набор данных с последовательным, индексно-последовательным или с прямым доступом (по физическому адресу). Возможен последовательный или произвольный доступ к записям (но не к полям). Поиск по многим ключам, как правило, не используется. Если используются иерархические файлы, то взаимосвязь «исходный — порожденный» программируется в прикладной программе;
- типовое программное обеспечение системы обработки данных представляет собой методы доступа, но не «управление данными». Данные в основном разрабатываются и оптимизируются для одного приложения;
- средства обеспечения защиты данных недостаточно надежны.

3.3.3. Первые системы управления данными (начало 1970-х гг.)

Для этого этапа характерно изменение представления о назначении и возможностях систем управления данными. По мере развития средств обработки данных становилось ясно, что прикладные программы желательно сделать независимыми не только от изменений в аппаратных средствах хранения, но также и от добавления к хранимым данным новых полей и новых взаимосвязей. Система должна быть способна обрабатывать новые типы запросов пользователей (рис. 3.4).

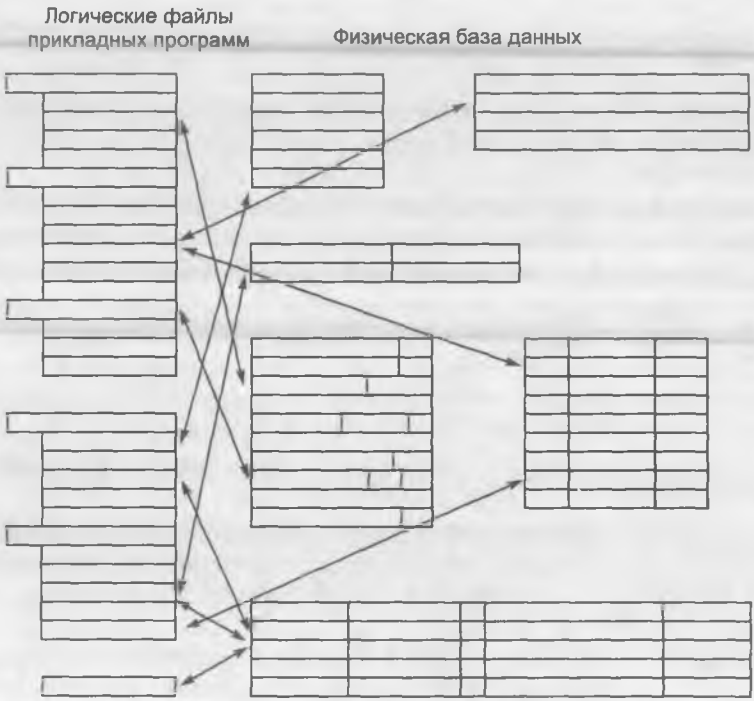


Рис. 3.4. Первые системы управления базами данных

Организация хранения и доступа в случае систем управления данными характеризуется следующими особенностями:

- различные логические файлы могут быть получены из одних и тех же физических данных. Доступ к одним и тем же данным

может осуществляться различными приложениями по различным путям, отвечающим требованиям этих приложений;

- данные адресуются на уровне полей и групп. Можно использовать поиск по многим ключам;
- физическая структура данных независима от прикладных программ. Ее можно изменять с целью повышения эффективности базы данных, не модифицируя при этом прикладные программы. Использование сложных форм организации данных не требует усложнения прикладных программ;
- элементы данных являются общими для различных приложений. Отсутствие избыточности способствует целостности данных.

3.3.4. Системы управления базами данных

Требования к СУБД основываются на том, что структура базы данных является менее *статичной*, чем файловая структура. СУБД должна обеспечивать два уровня независимости данных.

Из одних и тех же данных могут быть получены различные логические файлы, а доступ к одним и тем же данным со стороны различных приложений может осуществляться различными путями, отвечающими требованиям этих приложений.

Для этого вводятся два уровня независимости данных (рис. 3.5).

Логическая независимость данных означает, что общая логическая структура данных может быть изменена без изменения прикладных программ (изменение, конечно, не должно заключаться в удалении из базы данных таких элементов, которые используются прикладными программами).

Физическая независимость данных означает, что физическое расположение и организация данных могут изменяться, не вызывая при этом изменения ни общей логической структуры данных, ни прикладных программ.

Система обеспечивает привязку данных — связывание физического представления данных с программой, которая эти данные использует, путем преобразования обращения прикладной программы к *логической* записи или к ее элементам в машинные обращения к *физической* записи или ее элементам.

Физическая и логическая независимость данных обеспечивается программными средствами. Предусматривается существование глобального логического представления данных, а также использование

Рис. 3.5. Два уровня независимости данных

языка описания данных, *языка команд* для прикладного программиста и *языка запросов* для пользователя.

Для систем управления базами данных также характерны следующие особенности:

- так как базы данных конструируются для выдачи ответов на не запланированные заранее запросы, то используются дополнительные функционально-ориентированные структуры, например инвертированные файлы, позволяющие осуществлять быстрый поиск в базе данных по вторичным ключам;
- вводятся средства администрирования, которые позволяют управлять системой (в том числе защитой, секретностью, цело-

стностью и безопасностью данных), проектировать структуры, оптимальные для пользователей, обеспечивать импорт-экспорт и перемещение данных.

3.4. Схема управления данными в СУБД

Рассмотрим примерную последовательность операций, обеспечивающих чтение прикладной программой данных из базы данных, представленную на рис. 3.6:

(1) — прикладная программа (клиентское приложение) формирует и выдает системе управления базами данных запрос на чтение необходимых данных, содержащихся в базе;

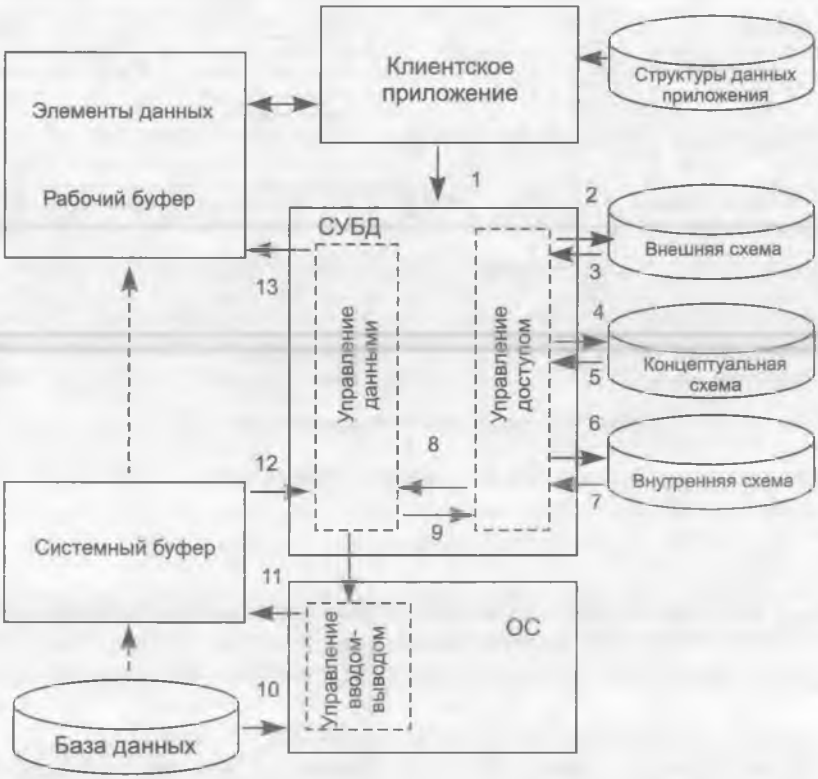


Рис. 3.6. Схема обработки запроса на выборку данных из БД

(2—3) — СУБД отыскивает описание затребованных данных в структуре описания данных прикладного уровня (внешняя схема);

(4—5) — СУБД по глобальному описанию БД (концептуальная схема) определяет необходимые данные на логическом уровне;

(6—7) — СУБД по описанию физической структуры БД (внутренняя схема) определяет физическую запись (или совокупность записей), которую необходимо считать для выборки данных, затребованных прикладной программой;

(8—9) — СУБД через подсистему управления потоками данных выдает операционной системе запрос на чтение хранимой записи;

(10—11) — подсистема управления вводом-выводом операционной системы осуществляет физическое чтение записи в системный буфер ОС;

(13) — СУБД выделяет необходимую логическую запись, осуществляет форматные преобразования, обусловленные различиями описаний на глобальном и прикладном уровнях, и передает для функциональной обработки приложением данные в рабочий буфер, выделяемый прикладной программой или самой СУБД.

3.5. Особенности и компромиссы реализаций баз данных

В заключение приведем основные отличительные особенности обработки данных, характерные для файловых систем и систем управления базами данных.

Файлы обладают следующими свойствами:

- файл, как правило, представляет собой совокупность записей одного типа, доступ к которым определяется типом *организации* файла и осуществляется только средствами операционной системы;
- файл описывают и используют в прикладной программе, работающей с данными.

Базы данных имеют следующие особенности:

- база данных представляет собой совокупность данных разного типа, причем часто по одним данным получают другие;
- база данных существует независимо от конкретной прикладной программы — база создается с целью интеграции данных, объединяющей данные многих приложений (но определенного на-

значения). База данных предназначена для *совместного, многофункционального* использования *многими* пользователями *один раз введенных* данных.

Надо отметить, что с точки зрения управления данными СУБД оперируют данными на содержательном уровне, хотя физические структуры, используемые для этих целей, могут и совпадать с аналогичными структурами, создаваемыми ОС.

Коренное же отличие СУБД от файловых систем ОС состоит в том, что СУБД устанавливает связь между *содержанием и адресом*, а ОС — между *именем и адресом* данных.

В то же время эта граница постоянно подвергается «атакам» с обеих сторон. Например, ОС-360 с «индексным доступом к данным», IN-PICK, включает язык поиска записей файлов по содержанию, UNIX — команды сортировки, коррекции или объединения содержимого текстовых файлов наподобие того, как это осуществляется с таблицами данных в СУБД. Тем не менее следует признать это скорее исключением, чем правилом, и в компетенцию ОС надо относить только связь «имя—адрес», оставляя другие зависимости на ответственность прикладных программ и оболочек СУБД и АИПС (автоматизированные информационно-поисковые системы).

Приемы распознавания программой элементов данных или записей относятся к такому типу взаимодействия программ и данных, когда описание данных размещено в программе, а файл данных организован в соответствии с этим описанием (рис. 3.7, *а*). Однако этот способ может привести к нарушению функционирования или разрушению данных, если из-за ошибок программиста или оператора к программе будет подсоединен «неправильный файл».

Для установления независимости программ от данных в некоторых системах описание данных размещают совместно с файлом данных (рис. 3.7, *б*). По такому принципу организован весьма распространенный *формат файла данных* (dbf-формат), происходящий от систем dBase — Clipper — Foxbase — FoxPro, а затем принятый и рядом других систем. В этом случае в начале файла создается заголовок, содержащий описание полей записи файла (имя, тип, длина данного, код информации и пр.), и, таким образом, описание данных файла в программе не нужно.

Недостатком такого подхода является, например, необходимость использования программистами тех же имен данных, что содержатся в описании файла.

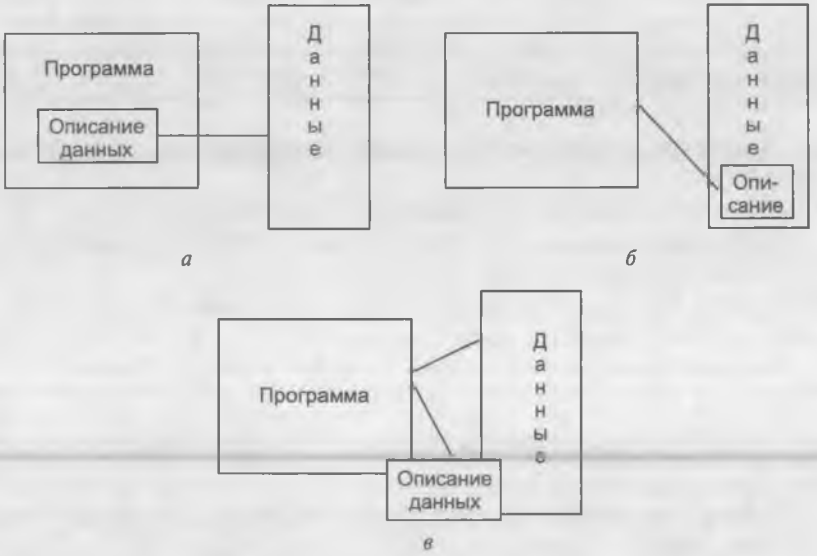


Рис. 3.7. Варианты размещения данных и их описания:
а — в прикладной программе; *б* — в файле данных; *в* — отдельным набором данных (словарь данных)

Следующим шагом явилось полное отделение описаний от данных и программ и сосредоточение их в специальных файлах (таблицах) — *словарях данных* (рис. 3.7, *в*), которые относятся к *базам данных и системам управления базами данных*.

В общем случае можно сказать, что основные задачи обработки данных, решаемые на основе концепций баз данных, сводятся к следующим вопросам.

1. Каким образом сложные нелинейные структуры данных представить в виде линейных — наиболее соответствующих принципу последовательного представления (хранения) в машинной памяти?

2. Каким образом организовать данные, чтобы была возможность эффективного добавления, удаления и редактирования данных?

3. Как организовать данные, чтобы использование пространства памяти (плотность данных) было достаточно рациональным, а скорость доступа к записям данных высокой?

4. Каким образом организовать данные, чтобы поиск был эффективным и позволял отыскивать записи по нескольким ключам?

При этом с точки зрения прагматики создание базы данных — это, по существу, попытка найти компромисс сразу по нескольким

направлениям и сочетаниям нескольких взаимообратных факторов (с точки зрения их влияния на показатель общей эффективности системы), в том числе следующих:

1) эффективность — простота;
2) скорость выборки — стоимость (сложность) аппаратных средств;
3) скорость выборки — сложность процедур доступа;
4) плотность данных — время доступа и сложность процедур;
5) независимость данных — производительность;
6) гибкость средств поиска — избыточность данных;
7) гибкость поиска — скорость поиска;
8) сложность процедур доступа — простота обслуживания.

Контрольные задания

1. Перечислите основные задачи обработки данных, решаемые на основе концепций баз данных.
2. Проведите сравнительный анализ понятий «физического» и «логического» представлений.
3. Определите соотношение физической и логической записей.
4. Приведите примерную схему организации файлового ввода-вывода.
5. Проведите сравнительный анализ процессов обработки данных средствами файловой системы и СУБД.
6. Перечислите основные этапы эволюции систем обработки данных.
7. Определите различия в концепциях обработки данных 1-го и 2-го этапов.
8. Определите различия в концепциях обработки данных 3-го и 4-го этапов.
9. Проведите сравнительный анализ объектов и функций, используемых в разных концепциях обработки данных.
10. Приведите схему управления данными в СУБД.

Глава 4
МОДЕЛИ И ЭТАПЫ ПРОЕКТИРОВАНИЯ БАЗ ДАННЫХ

4.1. Стадии проектирования и объекты моделирования

Не исключено, что у читателя создалось впечатление, будто мы уже владеем современной методологией или, по крайней мере, близки к этому, что, к сожалению, не так, и, может быть, мы никогда ничего подобного не добьемся. Всегда несложно охарактеризовать методологию на концептуальном уровне, весьма трудно применить ее на практике. Камень преткновения — сложность проникновения в существо предметной области (например, сложности понимания механизма деятельности организации) и адаптации ее к новым, возможно, лучшим условиям функционирования.

Аналогичные проблемы характерны и для СУБД в целом. Система баз данных должна стать органическим элементом системы управления организацией — вот залог ее успешного применения. Однако процесс ее внедрения связан с определенными изменениями в самой организации и в деятельности ее сотрудников, и мы всегда будем сталкиваться с естественной инертностью людей, когда речь идет о восприятии изменений...

Весьма важно, чтобы средства СУБД были адекватны потребностям пользователей. Поскольку разным пользователям могут понадобиться разные модели данных, языки данных и схемы, желательно, чтобы СУБД поддерживала множество средств, а пользователь мог выбирать из них наиболее подходящие. ...

Можно, конечно, поставить под сомнение ценность таких исследований. Действительно, каким бы плохим ни был язык программирования,

его, в конце концов, все-таки можно выучить. Точно так же и средства СУБД можно освоить за определенный период времени. Но проблема состоит не в освоении средств, а в эффективности их использования!

Следует отметить, что положения цитаты из [18], текст которой выделен выше курсивом, по-прежнему актуальны, хотя книга издана более четверти века назад. Действительно, средства проектирования непрерывно развиваются, но и задачи, решение которых пользователь предполагает автоматизировать с помощью систем баз данных, существенно усложнились.

С точки зрения объектов моделирования необходимо различать модели предметной области и модели базы данных. Эти модели взаимосвязаны, поскольку представляют собой образы одного и того же оригинала — некоторого множества предметов реального мира, информацию о которых предполагается хранить и обрабатывать с помощью проектируемой БД.

Модель предметной области ассоциируется с неформальным[1] уровнем семантического моделирования, а модель базы данных — с хорошо формализованным уровнем системы (и в частности, СУБД). В идеале целью семантического моделирования является формирование систематического основания для хорошо формализованного процесса проектирования базы данных. Здесь необходимо вспомнить следующие требования, предъявляемые к базам данных, и в частности к способам описания данных:

1) описания должны быть понятны пользователю, не проектировавшему базу;

2) однажды принятые способы представления данных должны допускать присоединение новых элементов данных без изменения существующих схем данных и прикладных программ;

3) СУБД должны позволять эффективно обрабатывать произвольные запросы к базе данных.

Эти требования отражают, с одной стороны, точку зрения пользователя, для которой характерны требования высокой степени общности и широты представления (или, по крайней мере, не громоздкость детальных описаний), позволяющих ему получить достаточно

[1] Правильнее было бы говорить о *неформализованности*, связанной с невозможностью обоснованного *однозначного* выбора (из реально существующих) как множества объектов, так и средств, используемых для моделирования (описания выделенных объектов).

сведений без затраты значительных временных или интеллектуальных ресурсов. С другой стороны — точку зрения администратора, выполняющего проектирование и оптимизацию базы данных, что предполагает высокую степень детализации и формализации, обеспечивающих обоснованность технических решений, а также возможность автоматизации проектирования.

Проектирование базы данных — это упорядоченный формализованный процесс создания *системы взаимосвязанных описаний,* т. е. таких моделей, которые связывают (фиксируют) хранимые данные с объектами предметной области, описываемыми этими данными. Прикладное назначение таких описаний состоит в том, чтобы пользователь, практически не имеющий представления об организации данных в БД (физическом размещении данных в памяти и механизмах их поиска), обращая запрос к базе, имел бы практическую возможность получить адекватную информацию о состоянии объекта предметной области.

Проектирование начинается с анализа предметной области и выявления функциональных и других требований к проектируемой системе. Подробнее этот процесс будет рассмотрен ниже, а здесь отметим, что проектирование обычно выполняется человеком (группой людей) — системным аналитиком (а на практике чаще администратором базы данных), которым может быть как специально выделенный сотрудник, так и будущий пользователь базы данных, достаточно хорошо знакомый с машинной обработкой данных.

Объединяя отдельные представления о содержимом базы данных, полученные в результате опроса пользователей, и свои представления о данных, которые могут потребоваться для решения практических задач, системный аналитик сначала создает обобщенное неформальное описание будущей базы данных. Это описание, выполненное с использованием естественного языка, математических выражений, таблиц, графов и других средств, понятных всем работающим над проектированием базы данных, называют *концептуальной* (или *инфологической*) *моделью.*

Такая человекоориентированная модель практически полностью независима от физических параметров среды хранения данных, которой может быть как память человека, так и ЭВМ. Поэтому концептуальная модель не изменяется до тех пор, пока изменения в реальном мире (той его части, которая отнесена к предметной области) не потребуют модификации соответствующего фрагмента описания для адекватного отражения предметной области.

Модели следующих стадий проектирования являются машинно ориентированными. С их помощью СУБД дает возможность программам и пользователям осуществлять доступ к хранимым данным лишь по их именам, не заботясь о физическом расположении этих данных.

Доступ к данным и манипулирование данными осуществляются в соответствии с моделью данных, которую поддерживают СУБД определенного класса (например, реляционные СУБД). Описание модели данных, создаваемое по концептуальной (инфологической) модели средствами языка описания данных, называют *логической* (или *даталогической*) *моделью*.

Для размещения и поиска данных на внешних запоминающих устройствах конкретной СУБД использует *физическую модель* данных.

Представленная трехуровневая архитектура (концептуальный, логический и физический уровни) позволяет обеспечить независимость хранимых данных от использующих их программ. Хранимые данные могут быть переписаны на другие носители, или может быть реорганизована их физическая структура, в том числе дополнена полями для новых приложений, но это повлечет лишь изменение физической и, возможно, логической модели данных. Главное, такие изменения физической и логической моделей не будут замечены существующими пользователями системы (окажутся «прозрачными» для них), так же как не будут замечены и вновь подключаемые пользователи. Кроме того, независимость данных обеспечивает возможность создания новых приложений для решения новых задач без разрушения существующих

В процессе развития теории систем баз данных термин «модель данных» имел разное содержание. Для более глубокого понимания существа отдельных понятий рассмотрим некоторые особенности использования этого понятия в контексте эволюции баз данных, представленные в [9].

О понятии «модель данных». Первоначально понятие модели данных употреблялось как синоним структуры данных в конкретной базе данных. Структурная трактовка полностью согласовывалась с математическим определением понятия модели как множества с заданными на нем отношениями. Но, следует отметить, что объектом моделирования в данном случае являются не данные вообще, а конкретная база данных. Разработки новых архитектурных подходов, основанных на идеях многоуровневой архитектуры СУБД, показали, что уже недостаточно рассматривать отображение представлений конкретной базы данных. Требовалось решение

на метауровне, позволяющее оперировать множествами всевозможных допустимых представлений баз данных в рамках заданной СУБД или, что эквивалентно, инструментальными средствами, используемыми для их спецификации. В этой связи возникла потребность в термине, который обозначал бы инструмент, а не результат моделирования и соответствовал бы, таким образом, множеству всевозможных баз данных некоторого класса. То есть инструмент моделирования баз данных должен включать не только средства структурирования данных, но и средства манипулирования данными. Поэтому модель данных в инструментальном смысле стала пониматься как алгебраическая система — множество всевозможных допустимых типов данных, а также определенных на них отношений и операций. Позднее в это понятие стали включать еще и ограничения целостности, которые могут налагаться на данные. В результате проблема отображения данных в многоуровневых СУБД и системах распределенных баз данных стала рассматриваться как проблема отображения моделей данных.

Важно подчеркнуть, что для разработчиков и пользователей СУБД точным определением реализованной в ней модели данных фактически являются средства языка определения данных и манипулирования данными. Поэтому отождествлять такой язык со схемой базы данных (результатом моделирования) — конкретной спецификацией в этом языке — неправомерно.

Начиная с середины 1970-х годов под влиянием предложенной в тот период концепции *абстрактных типов* само понятие типа данных в языках программирования стало трансформироваться таким образом, что в него стали вкладывать не только структурные свойства, но и элементы поведения (изменения данных). В дальнейшем это послужило основой для формирования концепции объекта, на которой базируются современные объектные модели.

В связи с этим был предложен новый подход, при котором модель данных рассматривается как система типов. Такой подход обеспечивал естественные возможности интеграции баз данных и языков программирования, способствовал формированию направления, связанного с созданием так называемых систем программирования баз данных. Трактовке модели данных как системы типов соответствуют не только уже существующие широко используемые модели, но также объектные модели, завоевывающие все большее влияние.

О развитии и конкурировании моделей. По аналогии с ситуацией 1970-х годов, когда велись споры о преимуществах сетевой, иерархической и реляционной модели данных (как известно, завершившиеся «ничейным» исходом открытой дискуссии Ч. Бахмана и Э. Кодда на Конференции ACM SIGMOD в 1975 г.), в настоящее время сформировалась новая тройка конкурентов — реляционная, объектная и многомерная модели.

Хотя, строго говоря, здесь речь идет лишь о двух моделях. Действительно, многомерные модели, коммерческие реализации которых появились в начале 1990-х годов для поддержки технологий OLAP, не основаны на каких-либо радикально новых идеях. Они представляют собой некоторое расширение активно исследовавшейся в 1970—1980-х гг. модели универсальных отношений новыми операционными возможностями, обеспечивающими, в частности, необходимые для OLAP функции агрегирования данных. Таким образом, многомерные модели представляют собой особую разновидность реляционной модели.

Многомерные модели — это «полноправные» модели данных. Так же как и другие модели, они используются для описания базы данных, определяя тем самым соответствующее ее природе представление данных, и предоставляют средства манипулирования данными. И в этом смысле они ничем не отличаются по своей функциональности от прочих моделей данных. В сегодняшних массовых реляционных технологиях многомерные модели ассоциируются с внешним уровнем архитектуры систем баз данных. Именно этим определяется их направленность на конечного пользователя. Нужно заметить, что обеспечение комфортных условий для работы конечного пользователя было также основной целью разработки модели универсального отношения.

Однако главная проблема в области массовых технологий заключается не столько в том, чтобы найти достойного преемника реляционной модели, сколько в том, что огромный балласт унаследованных систем (сегодня — уже реляционных) не дает возможности для резких технологических сдвигов и радикального обновления существующих технологий. Стремление избежать огромных кратковременных капиталовложений в случае перехода к принципиально новым технологиям приводит к необходимости лишь эволюционного пути развития. Отсюда рождение таких гибридов, как объектно-реляционные базы данных «нового поколения».

Документальные системы и интеграция моделей. Приведенные выше положения разрабатывались и действительно широко используются для баз данных хорошо структурированной информации. Однако уже сегодня одной из важнейших проблем становится обеспечение интеграции неоднородных информационных ресурсов, и в частности слабоструктурированных данных. Необходимость ее решения связывается со стремлением к полноценной интеграции систем баз данных в среду Web-технологий. При этом уже недостаточно простого обеспечения доступа к базе данных традиционным способом «из-под» HTML-форм. Нужна интеграция на модельном уровне. И не просто синтаксическая интеграция. В этом случае проблема семантической интероперабельности информационных ресурсов сводится к задаче разработки средств и технологий, предусматривающих явную спецификацию метаданных для ресурсов слабо структурированных данных на основе традиционных технологий моделирования из области баз данных. (Напомним, что в области систем баз данных анало-

гичная ситуация была преодолена благодаря предложениям CODASYL о необходимости явной спецификации схемы базы данных, независимой от приложений).

Именно на достижение этой цели направлены интенсивные разработки WWW-консорциумом языка XML и его инфраструктуры (фактически новой модели данных для этой среды), объектной модели документов и других средств, которые, как можно ожидать, в близкое время станут основой технологий управления информационными ресурсами. Это направление связано с другой глобальной проблемой — организацией распределенных неоднородных информационных ресурсов на основе построения репозиториев[1] метаданных, обеспечивающих возможность семантического отождествления ресурсов и, таким образом, возможность их целенаправленного использования.

Рассмотрим основные стадии процесса[2] моделирования, представленные на рис. 4.1.

4.2. Системный анализ предметной области

Базы данных сами по себе представляют относительную ценность. Базы данных — это всегда важнейшая, но только *одна из* составляющих некоторой информационной системы. Надо отметить, что любая ИС, предназначенная, например, для оперативного управления предприятием или архивного хранения и поиска документов, — это не только программы, данные и коммуникации, но также и люди (заказчики, пользователи, аналитики, разработчики), организационные структуры, а также цели, стимулы работы предприятия или отдельных людей. И все эти компоненты должны быть понятны как проектировщику, так и пользователю, а кроме того, непротиворечивым образом соединены в единую систему.

Главная идея процесса такого согласования состоит в том, что его надо начинать с анализа основных характеристик предметной области, рассматривая самые главные содержательные аспекты. И проводить его не «мысленно» и не «на словах», а на явно изложенных опи-

[1] Этому понятию в классических работах по проектированию баз данных соответствует понятие *словарь данных*.

[2] Здесь не рассматриваются работы, относящиеся к другим этапам жизненного цикла баз данных, такие как определение процедур заполнения и сопровождения БД, разработка конкретных программ обработки данных, развитие и улучшение структуры БД и т. д.

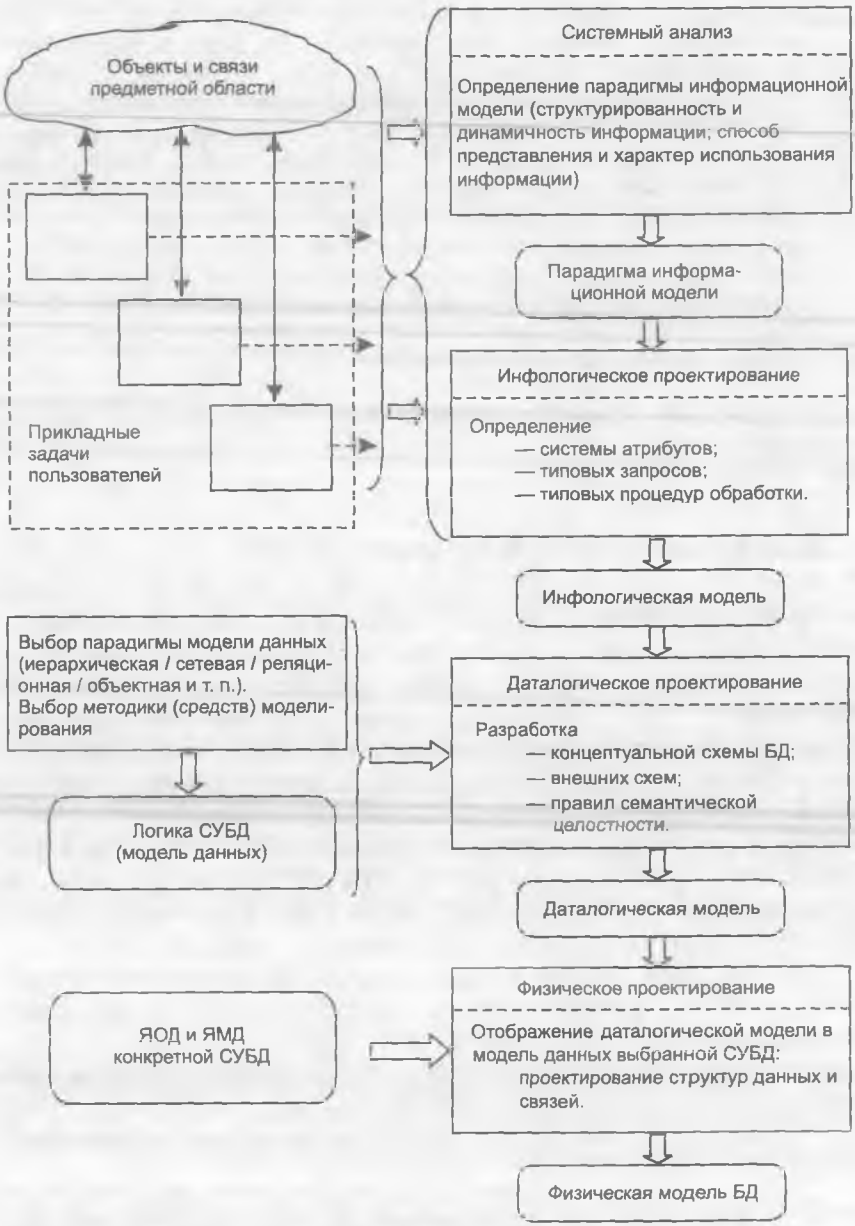


Рис. 4.1. Стадии и объекты процесса проектирования

саниях (моделях) объектов предметной области, позволяющих видеть все существенные взаимосвязи. Однако следует отметить, что попытки использования привычных нотаций формальных моделей (структурных, объектных или каких-либо других) на этом этапе приводят к более низкому (более детальному и в то же время ограниченному) уровню представления предметной области, чем это необходимо для общего понимания.

В общем случае рассматривают два подхода к определению состава и структуры предметной области.

Функциональный подход предполагает, что проектирование начинается с анализа задач и, соответственно, функций, обеспечивающих реализацию информационных потребностей.

При *объектном* подходе (акцентирующем внимание на предметах обработки) информационные потребности пользователей (задачи) жестко не фиксируются, а основное внимание сосредотачивается на выделении существенных объектов и связей, информация о которых может быть использована в прикладных задачах пользователя.

Условность такого деления достаточно очевидна, поэтому на практике используются компромиссные варианты, предполагающие по мере развития системы расширение как состава объектов, так и спектра прикладных задач.

В [27] была предложена простая, но концептуально мощная схема, показывающая различные уровни представления архитектуры ИС, различные виды ее «обеспечения», а также их основные взаимосвязи. На рис. 4.2 показана таблица, представленная в [6], аналогичная схеме Захмана. Три столбца отражают три раздела обеспечения системы: информационный (ДАННЫЕ), функциональный (ФУНКЦИИ) и коммуникационный (СЕТЬ).

Строки таблицы обозначают шесть уровней представления системы: реальная среда приложений, концептуальная модель, логическая модель, физическая модель, детальная реализация процедур, представления пользователя.

Полезность такой схемы состоит в том, что она помогает рассматривать задачи проекта в полном объеме, упорядочивать состав и структуру требований к системе, определять и фиксировать причинно-следственные связи.

Позднее появилось развитие такой «плоской» модели. В [26] рассматривалась представленная на рис. 4.3 схема, которая включает три новых столбца, отражающих еще три раздела: побудительные причи-

Рис. 4.2. Модель информационной системы Захмана

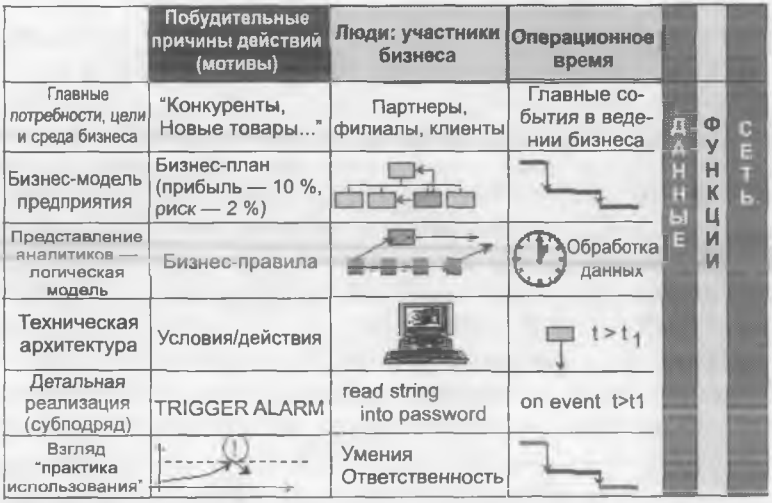


Рис. 4.3. Развитие модели информационной системы Захмана

ны действий системы, события и графики выполнения действий, а также «действующие лица» — люди и организационные структуры.

В результате появилось шесть разделов, которые содержат «ответы на вопросы»: почему выполняются действия, когда выполняются и

кто их выполняет, а также что делает система, как делает и где. При этом уровни представления (строки таблицы) остались те же.

Такое расширение позволило рассматривать потребности в контексте информационных технологий, соединять предметы и действия с человеческим фактором и операционной динамикой процессов.

Цель системного анализа предметной области как этапа проектирования — *выделить предметную область* как *систему объектов и их взаимосвязей,* определив при этом *функционально-информационные требования к их последующему представлению в виде системы взаимосвязанных данных.*

Главным результатом этапа системного анализа является определение *парадигмы* информационной (концептуальной) модели: требования к средствам представления системы определяются на основании анализа уровня структурированности информации и характера восприятия ее семантики пользователем (точная/приблизительная, четкая/неопределенная). Например, выбор *атрибутивной формы представления* объектов предметной области приведет, соответственно, к выбору парадигмы *фактографических баз данных*, а *вербальной* — к необходимости выбора *документальных БД.*

В дальнейшем изложении модели и средства проектирования мы будем рассматривать только для случая фактографических баз данных, использующих реляционную модель. В этом случае полученный результат — концептуальная модель предметной области — затем на логическом уровне преобразуется в реляционную модель. При таком предопределенном (с точки зрения логики представления и обработки данных) процессе проектирования более подходящими надо считать названия концептуального и логического этапов проектирования как, соответственно, *инфологический* и *даталогический.*

4.3. Концептуальные модели

Следующей за системным анализом стадией проектирования базы данных является построение *семантической модели* предметной области, которое базируется на анализе свойств и природы объектов предметной области и информационных потребностей будущих пользователей разрабатываемой системы. Эту стадию принято называть *концептуальным* проектированием, а ее результат — *концептуальной моделью* предметной области (объектом моделирования здесь является предметная область будущей системы). Этой стадии соответствуют

также ранее упомянутые термины «*инфологическое проектирование*» и «*инфологическая модель*».

Такие модели обобщенно представляют информационные потребности пользователей[1] и по существу являются средством коммуникации как для разработчиков, так и для пользователей на разных стадиях жизненного цикла базы данных.

Назначение концептуальных моделей определяет и некоторые специфические требования к средствам их представления. Помимо упомянутой независимости от среды реализации и требования адекватности отражения предметной области отметим следующие:

- формализованность, обеспечивающую возможность автоматизированной обработки и в том числе, например, автоматический контроль непротиворечивости;
- дружественность, обеспечивающую возможность использования наглядных графических средств отображения и обработки.

К концептуальным моделям относятся описания, по-разному и разными средствами отражающие предметную область. Помимо наиболее известного представления объектов и связей между ними (модель «сущность—связь») к концептуальному уровню описания предметной области можно отнести: логические (алгоритмические) связи между показателями; лингвистические свойства языка (синонимию, синтаксис и т. д.), используемого для вербального представления объектов; описание информационных потребностей пользователей в виде типовых запросов, отражающих процедурные особенности обращения к данным и т. п.

4.4. Логические модели

Задачей следующей стадии проектирования базы данных является выбор подходящего (по поддерживаемой модели данных) типа СУБД и отображение на логический уровень спецификаций концептуальной модели предметной области. Эту стадию называют логическим (или даталогическим) проектированием базы данных, а ее результатом является логическая модель (или концептуальная схема)

[1] Еще раз отметим, что модели ПрО, в отличие от моделей данных, используемых в качестве инструмента моделирования конкретных баз данных, не обязательно поддерживаются механизмами используемой СУБД, хотя это и может иметь место на практике.

базы данных, включающая определение всех информационных элементов (единиц) и связей, в том числе задание типов, характеристик и имен.

Хотя логическое проектирование оперирует не физическими записями, а логическими понятиями, связанными со структурой базы данных, тем не менее, особенности представления данных, языки агрегирования и манипулирования данными имеют определяющее влияние. Не все виды связей (например, «многие ко многим») могут быть непосредственно отображены в логической модели.

Кроме того, может быть много вариантов отображения концептуальной модели предметной области в логическую модель базы данных. Здесь следует учитывать влияние двух следующих значимых факторов, связанных с практикой разработки базы данных.

Во-первых, связи предметной области могут отображаться двумя путями: как декларативным — в логической схеме, так и процедурным — отработкой связей через программные модули, обрабатывающие (связывающие) соответствующие хранимые данные.

Во-вторых, существенным фактором может оказаться характер обработки информации. Например, частые обращения к совместно обрабатываемым данным очевидно предполагают их совместное хранение; а данные (особенно большой размерности), к которым обращаются редко, целесообразно хранить отдельно от часто используемых.

4.5. Физические модели

Стадия физического проектирования состоит в преобразовании логической модели в физическую при реализации базы данных средствами *конкретной* СУБД и в общем случае включает:

- выбор способа организации базы данных;
- разработку спецификации внутренней схемы БД средствами модели данных внутреннего уровня;
- описание отображения концептуальной схемы, создаваемой на этапе логического проектирования, во внутреннюю.

Важно заметить, что в отличие от ранних СУБД, многие современные системы не предоставляют разработчику какого-либо выбора на этой стадии. Реально к вопросам проектирования физической модели можно отнести выбор схемы размещения данных (разделение по файлам или тип RAID-массива) и определение числа и типа индексов (например, кластерный или некластерный).

Способ хранения базы данных определяется механизмами СУБД автоматически «по умолчанию» на основе спецификаций концептуальной схемы, реализованных с учетом особенностей языка описания данных конкретной СУБД, и внутренняя схема в явном виде в таких системах не используется.

Следует также отметить, что внешние схемы базы данных обычно конструируются на стадии разработки приложений.

4.6. Подходы к проектированию базы данных

Можно выделить два основных подхода к проектированию баз данных: *нисходящий* и *восходящий*.

При *восходящем* подходе этап системного анализа ориентирован на самый нижний уровень описания предметной области — представление ее в виде совокупности атрибутов (характеристических свойств). Следующий этап — выявление зависимостей, существующих между атрибутами в ПрО, и декомпозиция множества атрибутов на подмножества, характеризующие собственно типы объектов и связей. Таким образом, выделение объектов может происходить в соответствии с набором формальных правил. В главе 8 будет рассмотрен процесс нормализации отношений в рамках реляционной модели данных, который представляет собой один из вариантов восходящего подхода при проектировании баз данных.

Восходящий подход в случае сложных ПрО (например, характеризующихся большим количеством атрибутов, установить между которыми все существующие зависимости затруднительно), на самом деле будет очень неудобен для проектировщика. На начальных стадиях формулирования требований к данным бывает трудно также выявить и все необходимые атрибуты. Более того, здесь проявляется *ограниченность реляционной модели*, в частности:

- реляционная модель не предоставляет достаточных средств для фиксации смысла данных, т. е. семантика предметной области не фиксируется непосредственно в отношениях;
- для многих приложений трудно моделировать предметную область на основе плоских таблиц;
- хотя весь процесс проектирования происходит на основе учета зависимостей, реляционная модель не имеет средств представления (отражения семантики) этих зависимостей;

- несмотря на то что процесс проектирования начинается с выделения некоторых существенных для приложения объектов предметной области и выявления связей между ними, реляционная модель данных не предлагает какого-либо аппарата для различения объектов и связей.

Нисходящий подход к проектированию предопределяет приоритетность разработки концептуальной модели ПрО и является более приемлемым в случае сложных баз данных. Концептуальная модель призвана определить и зафиксировать основные типы объектов и связей, которые впоследствии могут уточняться до тех пор, пока не будет полностью отражена специфика задач конкретной ПрО.

Применяется также и *смешанная стратегия проектирования*, при которой для различных фрагментов ПрО используются восходящий или нисходящий подходы, после чего полученные фрагменты собираются в общую модель

4.7. Средства автоматизации проектирования

Формализованные знания о предметной области в общем случае могут быть представлены в виде текстовых описаний: наборов должностных инструкций, правил ведения дел и т. п. Однако текстовый способ представления модели предметной области не эффективен. Более информативными и полезными при разработке баз данных и информационных систем являются описания предметной области, выполненные при помощи специализированных графических нотаций, реализующих методики представления знаний о предметной области. Наиболее известными на сегодняшний день являются методика структурного анализа SADT (Structured Analysis and Design Technique) и основанная на ней нотация IDEF0, диаграммы потоков данных, методика объектно ориентированного анализа UML (Unified Modeling Language) и др. Любая из этих моделей описывает, с одной стороны, процессы, происходящие в предметной области, а с другой — данные, используемые этими процессами.

Наиболее полная система моделей, на которую опираются методики функционального, информационного и поведенческого моделирования ПрО, представлена в семействе стандартов IDEF (Integrated DEFinition). В состав семейства входят следующие IDEF-модели (табл. 4.1).

Методология концептуального проектирования, основанная на наглядной графической технике, предоставила в распоряжение разра-

Таблица 4.1

Название	Назначение
IDEF0	Реализует методику функционального моделирования сложных систем. Рекомендуется для начальных стадий проектирования систем управления, производства, бизнеса и т. п., объединяющих людей, оборудование, программное обеспечение. Позволяет формулировать ответы на вопрос: «Что делает система?»
IDEF1 и IDEF1X	Реализуют методики стадии инфологического проектирования. IDEF1X имеет формальный язык диаграмм «сущность—связь» (ERD — Entity—Relations Diagrams), предназначенный для описания объектов и связей (отношений) между ними. Разработка модели инфологического уровня IDEF1X выполняется в несколько этапов: • выяснение цели проекта, составление плана сбора информации (обычно исходные положения следуют из модели IDEF0); • определение основных сущностей — объектов базы данных; • определение ключевых и не ключевых атрибутов сущностей; • выявление и определение основных связей (отношений) между объектами; • представление результатов в графической форме (в виде ER-диаграммы)
IDEF2 и IDEF3	Реализуют поведенческое моделирование. Позволяют формулировать ответы на вопросы: «Как система выполняет функцию?». Предоставляют инструментарий для наглядного представления и моделирования сценариев автоматизируемых процессов
IDEF4	Реализует объектно ориентированный анализ больших систем. Предоставляет пользователю графический язык для изображения классов, диаграмм наследования, полиморфизма методов
IDEF5	Реализует представление онтологической информации в удобной для пользователя форме. Характерной чертой онтологического анализа является разделение реального мира на классы объектов и определение их онтологий — совокупности фундаментальных свойств, которые определяют их изменение и поведение. Для поддержания процесса построения онтологий существуют специальные онтологические языки, включающие символические обозначения объектов, их ассоциаций, схемы описания отношений классификации («часть—целое» и т. п.)
IDEF6	Реализует сохранение рационального опыта проектирования ИС, что способствует предотвращению структурных ошибок
IDEF8	Реализует проектирование диалогов «человек — техническая система»

Окончание табл. 4.1

Название	Назначение
IDEF9	Реализует анализ существующих условий и ограничений (физических, политических, юридических и т. п.) и их влияние на принимаемые решения в процессе реинжиниринга
IDEF14	Реализует представление и анализ данных при проектировании вычислительных сетей на графическом языке с описанием конфигураций, очередей, сетевых компонентов, требований надежности и т. п.

ботчиков информационных систем строгие формализованные методы описания ИС и принимаемых технических решений. Эти модели по существу представляют собой систему соглашений, обеспечивающих взаимопонимание бизнес-аналитика, представляющего реалии предметной области, и программиста (или программного средства), создающего модель данных для отражения состояния этой ПрО. Если соглашения в точности будут реализованы в программных продуктах, основанных на этой методологии, то такая автоматизированная система, умеющая «читать» разработанные аналитиком модели, позволит контролировать синтаксис модели и в итоге сгенерировать схему данных.

Вслед за методологией концептуального проектирования появились специализированные программно-технологические средства специального класса — CASE-средства, реализующие технологию создания и сопровождения ИС.

CASE-технология представляет собой набор инструментальных средств, позволяющих в наглядной форме моделировать предметную область, анализировать эту модель на всех этапах разработки и сопровождения ИС и разрабатывать приложения в соответствии с информационными потребностями пользователей.

Термин CASE расшифровывается как «Computer-Assisted Software Engineering» или «Computer-Aided Software/System Engineering». Фактически CASE-средства представляют собой новый тип графически ориентированных инструментов поддержки жизненного цикла (ЖЦ) программных систем. Обычно к ним относят любое программное средство, обеспечивающее автоматическую помощь при проектировании, разработке, сопровождении программной системы и имеющее следующие дополнительные черты:

- *развитые графические возможности,* используемые для описания и документирования программной системы;

- *интеграция*, обеспечивающая легкость передачи данных и позволяющая управлять всем процессом проектирования и разработки;
- *использование компьютерного хранилища* (репозитория) для всей информации о проекте как основа для взаимодействия разработчиков, автоматического создания компонентов программных систем и повторного использования в будущих системах.

Назначение CASE-средств, представленных сегодня на мировом рынке десятками и сотнями продуктов, многообразно. В соответствии с функциональной ориентацией на те или иные процессы жизненного цикла ИС их можно отнести к следующим группам:

- построение и анализ моделей предметной области и интерфейсов (например, Design/IDEF, AllFusion Process Modeler, Silverrun, System Architect);
- проектирование баз данных и генерация их схем для основных СУБД (например, Oracle Designer, встроенные CASE-средства СУБД Microsoft SQL Server, AllFusion ERwin Data Modeler);
- разработка приложений и создание их программного кода (к ним относятся средства 4GL — Uniface, PowerBuilder, Developer/2000, SQL Windows, Delphi и др.);
- реинжиниринг процессов и баз данных (например, Rational Rose, Object Team) и др.

4.8. Типология моделей

Трехуровневая система моделей представления предметной области и базы данных иллюстрируется рис. 4.4.

Основные отличия любых методов представления информации заключаются в том, каким способом фиксируется семантика предметной области. Следует особо отметить, что для любого метода представления предметной области (в контексте создания и использования машинных баз данных) в основе формирования представления лежит *кодирование* понятий и отношений между понятиями.

Этап системного анализа, выявив степень структурированности и динамичности информации, а также способ представления и характер использования, определяет направление концептуального моделирования — выбор модели представления хорошо или слабо структурированной информации.

Рис. 4.4. Система моделей представления информации

Декомпозиция предметной области на совокупность объектов и связей, с одной стороны, и на предопределенное множество функций — с другой, хорошо описывается средствами ER-диаграмм (Entity Relationship Diagram), диаграмм потоков данных (Data Flow Diagram), функциональной моделью IDEF0 и др. Однако в случае, когда база данных хранит описания разнородных объектов (с разным набором атрибутов), взаимодействие которых изначально не определено, а значения атрибутов зависят от контекста использования, перечисленные модели не дают адекватного представления о предметной области.

Различаются и методы отображения, используемые на этапе построения логических моделей, отражающих способ идентификации

элементов и связей, но, что особенно важно, в контексте их будущего представления в одномерном пространстве памяти вычислительной машины. Модели подразделяются на фактографические — ориентированные на представление хорошо структурированной информации, и документальные — представляющие наиболее распространенный способ отражения слабо структурированной информации. Если в первом случае говорят о реляционной, иерархической или сетевой моделях данных, то во втором — об инвертированной и прямой организации документальных массивов.

Разделение баз данных на фактографические и документальные в этой группе моделей является достаточно условным. Документ, как последовательность полей может быть представлен, в частности, и реляционной моделью. И в этом случае выбор специализированного решения чаще всего обусловливается требованием общей эффективности.

Представленная здесь типология моделей не претендует на полноту и не является классификацией в точном смысле этого слова. Она, скорее, иллюстрирует эклектичность преобладающих в разное время взглядов, методов и решений, используемых при проектировании и реализации баз данных.

Контрольные вопросы

1. Перечислите основные этапы проектирования БД.
2. Перечислите задачи этапа системного анализа.
3. Определите понятие «парадигма моделирования».
4. Что называется инфологической моделью?
5. В чем состоит сходство и отличие даталогической и физической модели?
6. В чем заключается даталогическое проектирование?
7. Охарактеризуйте модель информационной системы Захмана.
8. Охарактеризуйте понятие «модель данных».
9. Приведите классификационную схему моделей БД.
10. В чем разница между фактографическими и документальными БД?

Глава 5
КОНЦЕПТУАЛЬНОЕ МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

5.1. Анализ предметной области — определение информационных потребностей пользователей

Проектирование базы данных основано на сборе и анализе информации о той части реального мира, функции и задачи которой подлежат автоматизации. Совокупность методов сбора этой информации известна под общим названием *методики сбора фактов.* Приведем из [10] пять чаще всего используемых методов.

1. Изучение документации — бумажных и электронных форм, отчетов, обзоров, протоколов, распоряжений и т. п., связанных с решаемыми задачами.

2. Собеседование — интервьюирование будущих пользователей с целью выяснения требований, толкования фактов, сбора мнений и предложений и т. п.; различают не структурированные (свободные, не имеющие конкретной направленности) и структурированные (содержащие перечень заранее подготовленных вопросов и поэтому более эффективные) интервью.

3. Наблюдение за работой предполагаемых пользователей — активное или пассивное участие в выполнении повседневных функций; наиболее эффективно в случае, когда конечные пользователи не могут точно объяснить суть возникающих проблем.

4. Анкетирование — проведение опросов с помощью специально подготовленных хорошо структурированных анкет, позволяющих получить сведения от большого количества респондентов (вопросы при этом могут быть как в фиксированной, так и в свободной форме).

5. Проведение исследований — поиск информации об аналогичных решениях.

Анализ собранной информации должен проводиться с учетом потребностей не только сегодняшних пользователей и текущих задач проектируемой системы, но и возможного расширения или развития предметной области, присоединения новых пользователей и решения новых задач.

Этап первоначального сбора информации предполагает выявление всех пользователей и группировку их по принципу общности информационных потребностей и решаемых задач. Для каждой группы пользователей должно быть определено свое *пользовательское представление* — совокупность необходимых для работы данных и действий над ними. При этом требования разных пользовательских представлений могут частично пересекаться или полностью не совпадать.

Пользовательское представление (как минимум) должно содержать:

- описание используемых или формируемых данных;
- подробные сведения о способах использования или получения (формирования) данных.

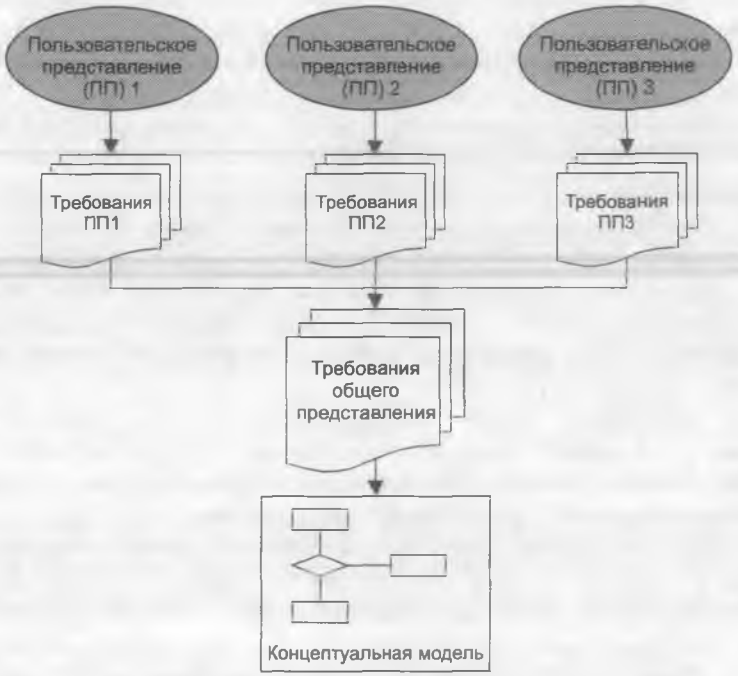


Рис. 5.1. Централизованный подход

Для определения общих требований к базе данных с несколькими пользовательскими представлениями применяют один их следующих методов.

Метод централизованного подхода предполагает предварительное объединение требований различных пользовательских представлений в единый набор, именуемый *общим представлением*. Общее представление далее рассматривается как основа для построения концептуальной модели БД (рис. 5.1). Такой подход наиболее конструктивен в случае, когда требования различных пользовательских представлений в значительной степени пересекаются.

Метод интеграции представлений основывается на построении локальных концептуальных моделей для каждого пользовательского представления. На одном из последующих этапов проектирования БД локальные модели объединяются для создания глобальной модели, которая отвечает требованиям всех пользовательских представлений (рис. 5.2). Такой подход, как правило, предпочтительнее, если между

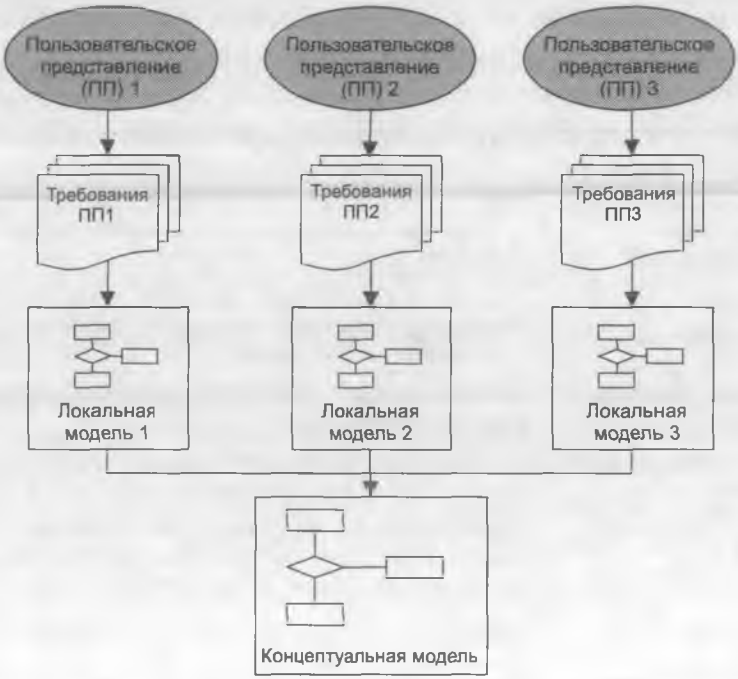


Рис. 5.2. Интеграция представлений

пользовательскими представлениями имеются существенные различия и работа по созданию функциональных интерфейсов может вестись в разных направлениях.

5.2. Критерии оценки концептуальной модели и проверка на адекватность

Сбор и анализ требований является предварительным этапом концептуального проектирования базы данных: собранная на этом этапе информация может быть плохо структурирована и содержать сведения, которые впоследствии потребуется более четко формализовать.

Формализованные требования к БД и различным приложениям представляются в виде моделей концептуального уровня проектирования и изображаются с помощью нотаций IDEF-моделей, диаграмм потоков данных (Data Flow Diagrams — DFD), диаграмм классов и прецедентов UML и т. п.

Оптимальная модель концептуального уровня проектирования должна удовлетворять критериям, перечисленным в табл. 5.1. Следует отметить, что на практике иногда эти критерии оказываются несо-

Таблица 5.1. **Критерии оценки концептуальной модели**

Критерий	Описание
Структурная достоверность	Соответствие способу определения и организации информации
Простота	Удобство работы с моделью как для профессионалов в области разработки информационных систем и БД, так и для обычных пользователей
Выразительность	Способность представлять связи между данными, различия и ограничения.
Отсутствие избыточности	Исключение излишней информации (т. е. любая часть данных должна быть представлена один раз)
Универсальность	Отсутствие принадлежности к какой-либо особой технологии и, как следствие, возможность использования во многих приложениях и технологиях
Расширяемость	Способность развиваться и включать новые требования
Целостность	Согласованность со способами использования и управления информацией

вместимыми: например, требование наибольшей *выразительности* модели может войти в противоречие с ее *простотой*.

Для проверки концептуальной модели на адекватность необходимо:

- проверить модель на отсутствие избыточности;
- проверить соответствие каждой локальной концептуальной модели конкретным пользовательским функциям;
- обсудить локальные концептуальные модели с конечными пользователями.

5.3. Модель «Сущность — связь»

Наиболее распространенным средством моделирования предметной области систем, ориентированных на обработку фактографической информации, является модель «Сущность — связь» (Entity-Relationship Model — ERM), впервые предложенная Питером Пин-Шэн Ченом в 1976 г. Эта модель традиционно используется в структурном анализе и проектировании, но, по существу, реализует объектный подход к моделированию предметной области.

Модель «Сущность — связь» положена в основу значительного количества CASE-средств, поддерживающих полный цикл разработки баз данных или отдельные его этапы. При этом многие из них не только поддерживают этап концептуального проектирования предметной области, но и позволяют осуществить на основе построенной концептуальной модели логическое проектирование путем автоматической генерации концептуальной схемы базы данных для выбранной СУБД.

Моделирование предметной области в этом случае базируется на использовании графических диаграмм, включающих сравнительно небольшое число компонентов и, самое важное, — *технологию построения* таких диаграмм.

Семантическую основу ER-модели составляют следующие предположения:

- та часть реального мира (совокупность взаимосвязанных объектов), сведения о которых должны быть помещены в базу данных, может быть *представлена* как совокупность *сущностей*;
- каждая сущность обладает характеристическими свойствами (атрибутами), отличающими ее от других сущностей и позволяющими ее *идентифицировать*;

- сущности можно классифицировать по типам сущностей: каждый экземпляр сущности (представляющий некоторый объект) может быть отнесен классу — *типу сущностей*, каждый экземпляр которого обладает общими для них свойствами и свойством, отличающим их от сущностей других классов;
- систематизация представления, основанная на классах, в общем случае предполагает иерархическую зависимость типов: сущность типа *A* является *подтипом* сущности *B*, если каждый экземпляр типа *A* является экземпляром сущности типа *B*;
- взаимосвязи объектов могут быть представлены как *связи* — сущности[1], которые служат для фиксирования (представления) взаимозависимости двух или нескольких сущностей.

Здесь следует еще раз подчеркнуть информационную природу понятия *сущность* и его соотношение с материальными или воображаемыми объектами предметной области. Любой объект предметной области обладает свойствами, часть из которых выделяется как характеристические — значимые с точки зрения прикладной задачи. При этом, например, в процессе анализа и систематизации предметной области обычно выделяются *классы* — совокупности объектов, обладающих одинаковым набором свойств, задаваемых в виде *наборов атрибутов* (значения атрибутов для объектов одного класса, естественно, могут различаться). Соответственно, на уровне представления предметной области (т. е. ее инфологической модели) объекту, рассматриваемому как понятие (объект в сознании человека), соответствует понятие *сущность*; объекту как части материального мира (и существующему независимо от сознания человека) соответствует понятие *экземпляр сущности*; классу объектов соответствует понятие *тип сущности*.

В дальнейшем, поскольку в инфологической модели рассматриваются не отдельные экземпляры объектов, а классы, мы не будем различать соответствующие понятия этих двух уровней, т. е. будем предполагать тождественность понятий *объект* и *сущность*, *свойство объекта* и *свойство сущности*.

ER-модель, как описание предметной области, должна определить объекты и взаимосвязи между ними, т. е. установить связи следующих двух типов: 1) связи между объектами и наборами характери-

[1] Такое определение связи, как сущности особого рода, отражает существо реляционного подхода, для которого характерно единообразное представление сущностей всех типов, включая связи, посредством переменных-отношений.

стических свойств (которые таким образом определяют сами объекты); 2) связи между объектами, задающие характер и функциональную природу их взаимозависимости.

5.3.1. Сущность и свойство сущности

Сущность, с помощью которой моделируется класс однотипных объектов, определяется в [18] как «предмет, который может быть четко идентифицирован». Так же как каждый объект уникально характеризуется набором значений свойств, сущность должна *определяться* таким *набором свойств*, который позволял бы различать отдельные экземпляры сущности. Каждый экземпляр сущности должен быть отличим от любого другого экземпляра той же сущности (это требование аналогично требованию отсутствия кортежей-дубликатов в реляционных таблицах). Например, для однозначной идентификации каждого экземпляра сущности «Сотрудник» вводится свойство «Табельный номер», которое вследствие своей природы будет всегда иметь уникальное значение в рамках предприятия. То есть уникальным идентификатором сущности может являться свойство, комбинация свойств, комбинация связей или комбинация связей и свойств, однозначно отличающая любой экземпляр сущности от других экземпляров сущности того же типа.

Сущность имеет *имя*, уникальное в пределах модели. При этом *имя сущности* — это *имя типа*, а не некоторого конкретного экземпляра.

Сущности подразделяются на *сильные* и *слабые*. Сущность является слабой, если ее существование в предметной области как самостоятельной невозможно и зависит от другой сущности — сильной по отношению к ней. Например, сущность «Документ» (паспорт, диплом, студенческий билет и т. п.) является слабой по отношению к сущности «Человек»: если будет удалена информация, соответствующая конкретной личности, то сведения о личных документах тоже должны быть удалены.

Типология характеристических *свойств* сущности аналогична типологии свойств объекта и приведена в п. 2.1.

Следует отметить, что для каждой сущности должно быть определено свойство (совокупность свойств) *первичного ключа* — уникального идентификатора, однозначно определяющего каждый отдельный экземпляр сущности.

Выбор и задание первичного ключа сущности должны подчиняться следующим правилам.

1. По возможности первичный ключ должен быть *наиболее компактным* из всех потенциальных ключей (вариантов уникальной идентификации), при этом предпочтительный тип данных для первичного ключа — целочисленный. Первичный ключ может быть составным, но увеличение количества атрибутов, входящих в него, противоречит требованию компактности.

2. Значения первичного ключа *не должны подвергаться частым модификациям* (в идеальном случае — вообще не должны меняться).

3. *Правила модификации первичного ключа должны контролироваться внутренней функциональностью* предметной области, а не решениями, которые принимаются за ее пределами. Например, в базе данных, разрабатываемой для нужд деканата, для сущности СТУДЕНТ не следует назначать первичным ключом такие обладающие уникальностью атрибуты, как серию и номер паспорта, так как их изменение может быть инициировано самим студеном, а не администрацией факультета.

4. Если среди информации, собранной о сущности, не удается выделить данные, претендующие на роль первичного ключа, то рекомендуется рассмотреть возможность создания *суррогатного первичного ключа,* который, не неся никакой семантической нагрузки, просто служит идентификатором конкретного экземпляра сущности.

5.3.2. Связи между сущностями

Кроме связей между сущностью и ее свойствами концептуальная модель отражает связи между самими сущностями. *Связь* в [18] определяется как «ассоциация, объединяющая несколько сущностей». Эта ассоциация всегда может существовать между разными сущностями или между сущностью и ей же самой (рекурсивная связь). Сущности, объединяемые связью, называются *участниками*. *Степень связи* определяется количеством участников связи[1].

Если каждый экземпляр сущности участвует по крайней мере в одном экземпляре связи, то такое участие этой сущности называется

[1] В большинстве CASE-систем принята упрощенная форма: эта ассоциация всегда *должна быть бинарной* и может существовать только между двумя разными сущностями или между сущностью и ей же самой.

полным (или *обязательным* — связь типа «должен»); в противном случае — *неполным* (или *необязательным* — связь типа «может»).

Как и сущность, связь является *типовым* понятием, т. е. все экземпляры связываемых сущностей подчиняются правилам связывания типов. Принципиальность различия связей между типами и экземплярами сущностей иллюстрирует рис. 5.3. Количественный характер участия экземпляров сущностей (один или многие) задается *типом связи* (или *мощностью связи*). Возможны следующие типы: «*один к одному*» (1 : 1) — рис. 5.3, *а*, «*один ко многим*» (1 : М) или «*мно-*

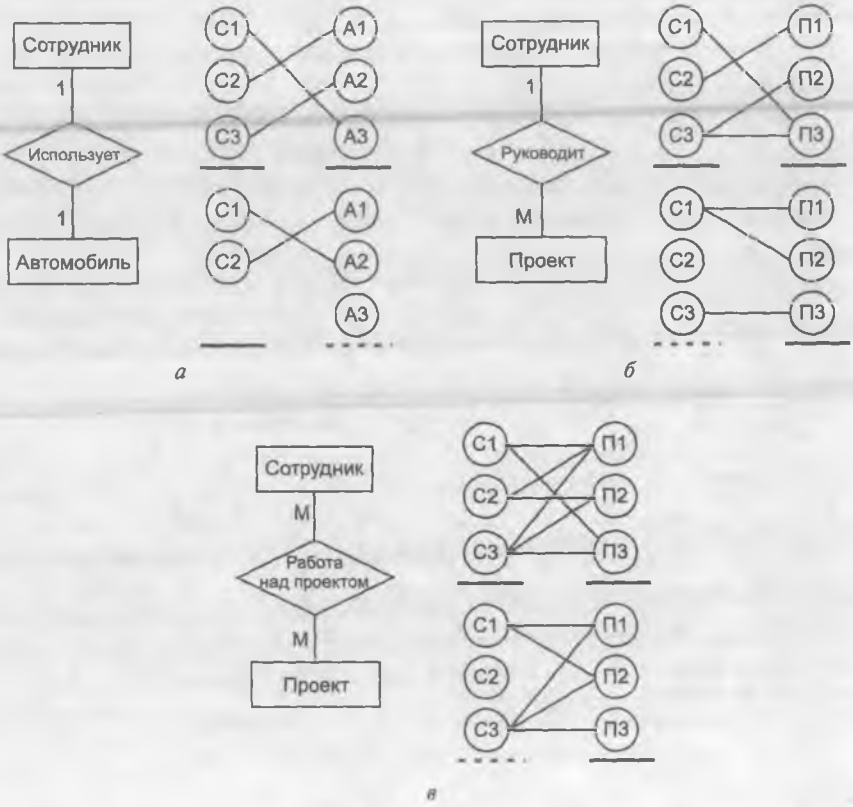


Рис. 5.3. Связи между типами и экземплярами сущностей:
а — связь мощности «один к одному» (1 : 1); *б* — связи мощности «один ко многим» (1 : М) и «многие к одному» (М : 1); *в* — связь мощности «многие ко многим» (М : М)

гие к одному» (M : 1) — рис. 5.3, *б*, «*многие ко многим*» (M:M)[1] — рис. 5.3, *в*. Сплошная линия на рисунках обозначает полное (или обязательное) участие сущности в связи, пунктирная — неполное (или необязательное).

5.3.3. Супертип и подтип

Сущность может быть расщеплена на два или более взаимно исключающих *подтипов*, каждый из которых включает общие атрибуты и/или связи. Эти общие атрибуты и/или связи явно определяются один раз на более высоком уровне. В подтипах могут определяться собственные атрибуты и/или связи. В принципе выделение подтипов может продолжаться на более низких уровнях, но в большинстве случаев оказывается достаточно двух-трех уровней.

Сущность, на основе которой определяются подтипы, называется *супертипом*. Подтипы образуют либо полное множество (т. е. любой экземпляр супертипа должен относиться к некоторому подтипу) либо неполное (могут существовать экземпляры супертипа, не относящиеся ни к одному из подтипов). Подтипы одного уровня могут допускать пересечение экземпляров — один экземпляр супертипа может принадлежать двум и более подтипам.

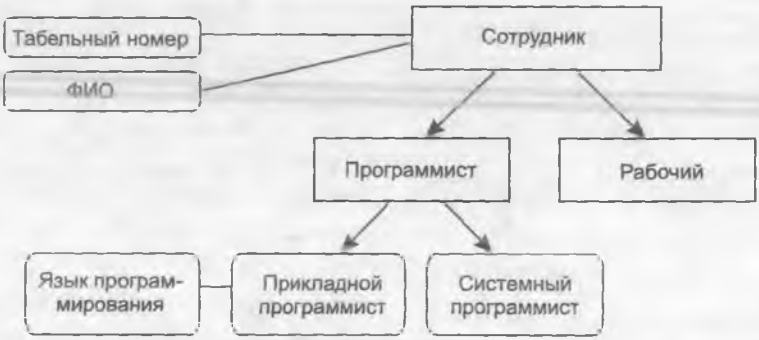


Рис. 5.4. Пример иерархии типов сущностей

[1] Согласно положениям реляционной модели «правильной» связью является только связь типа «*многие ко многим*», поскольку для ее реализации создается отдельная переменная-отношение. Связи «*один к одному*», «*один ко многим*» всегда могут быть представлены с помощью механизма внешнего ключа одной из переменных-отношений.

Подтип наследует свойства и связи супертипа. Например, тип сущности ПРОГРАММИСТ является подтипом сущности СОТРУДНИК. Программисты обладают всеми свойствами сотрудников и участвуют во всех связях, однако обратные утверждения неверны.

Тип сущности, его подтипы, подтипы этих подтипов и т. д. образуют *иерархию типов сущности*, пример которой приведен на рис. 5.4.

5.3.4. Нотации ER-диаграмм

Как было отмечено ранее, одна из основных целей семантического моделирования состоит в том, чтобы результаты анализа предметной области были отражены в простом, наглядном, но в то же время формализованном и достаточно информативном виде.

В этом смысле моделирование предметной области, базирующееся на использовании графических диаграмм, является удачным решением. Графическая форма позволяет отобразить в компактном виде (за счет наглядных условных обозначений) типологию и свойства сущностей и связей, а формализмы, положенные в основу ER-диаграмм, дают возможность сформулировать конечный набор правил проектирования логической структуры базы данных.

Диаграмма — графическое представление множества элементов, чаще всего изображаемое в виде связного графа с вершинами-объектами и ребрами-отношениями. На ER-диаграмме вершинами являются сущности и (например, в нотации Чена) свойства сущностей. Ребра ER-диаграммы представляют собой связи между сущностями и между сущностью и ее свойствами.

Существует несколько различных нотаций (языков) изображения ER-диаграмм. Исторически первой является нотация Чена; в семействе стандартов IDEF модель «сущность—связь» реализуется нотацией IDEF1X; используются нотации Мартина и Баркера, а также графические элементы UML.

Нотация UML

На рис. 5.5 приведен пример ER-диаграммы ПрО в нотации UML.

Унифицированный язык моделирования UML (Unified Modeling Language) представляет собой язык для определения, представления, проектирования и документирования программных систем, организационно-экономических систем, технических систем и других сис-

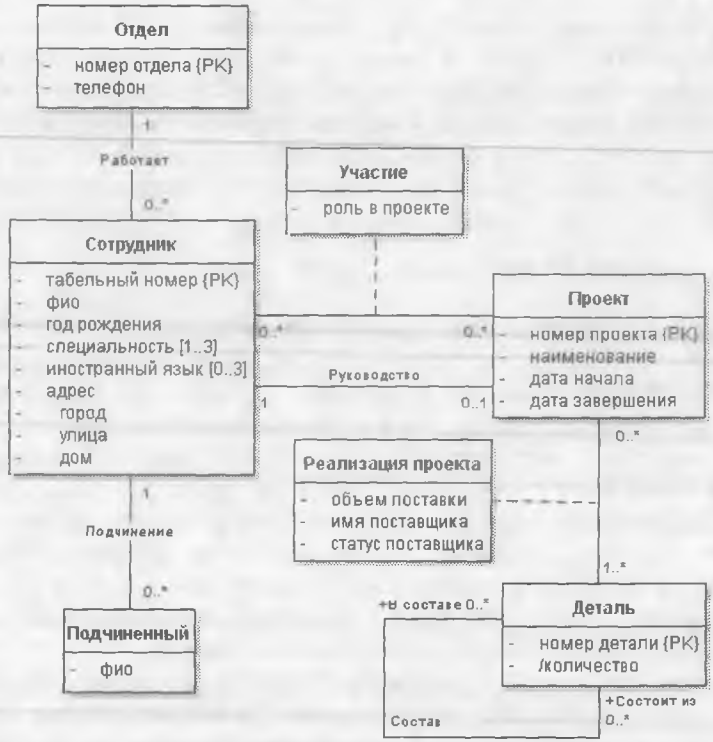


Рис. 5.5. ER-диаграмма в нотации UML

тем различной природы. Структуру UML составляет стандартный набор диаграмм и нотаций.

Главными в разработке UML были следующие цели:

- предоставить готовый к использованию выразительный язык визуального моделирования, позволяющий разрабатывать осмысленные модели и обмениваться ими;
- предусмотреть механизмы расширяемости базовых концепций языка;
- обеспечить независимость от конкретных языков программирования и процессов разработки;
- интегрировать лучший практический опыт.

Международный стандарт на UML (ISO\IEC 19501) был принят в 2005 г., однако развитие языка продолжается организацией по стандартизации в области объектно ориентированных методов и технологий (OMG — Object Management Group).

Рассмотрим правила изображения сущностей, свойств и связей в этой нотации.

Каждый тип сущности в ER-диаграммах нотации UML представляется в виде прямоугольника, содержащего имя сущности и перечень свойств сущности. В качестве имени сущности обычно используется существительное в единственном числе (например, ОТДЕЛ, ПРОЕКТ, СОТРУДНИК). Имя сущности указывается в верхней части прямоугольника с прописной буквы, имена свойств перечисляются в нижней части прямоугольника и начинаются со строчной буквы.

Слабые сущности характеризуются тем, что их нельзя однозначно идентифицировать только с помощью собственных атрибутов (в силу их зависимости от «родительских» сущностей и невозможности самостоятельного существования). На ER-диаграммах слабые сущности не имеют первичных ключей (например, сущность ПОДЧИНЕННЫЙ).

Свойства служат для уточнения, идентификации, характеристики или выражения состояния сущности или связи. Отражение на ER-диаграммах типологии свойств представлено в табл. 5.2.

Таблица 5.2. **Изображение типов свойств в UML**

Тип свойства	Описание	Пример
Свойство первичного ключа	После имени свойства в фигурных скобках помещается идентификация первичного ключа {PK}	**Сотрудник** - табельный номер {PK}
Многозначное	После имени свойства в квадратных скобках задаются возможные границы изменения количества значений [1..N]	**Сотрудник** - специальность [1..3]
Производное	Имя свойства начинается с наклонной черты «/»	**Деталь** - /количество
Условное (необязательное)	После имени свойства в квадратных скобках задаются возможные границы изменения количества значений [0..N]	**Сотрудник** - иностранный язык [0..*]
Составное	Под именем составного свойства перечисляются с отступом вправо имена простых свойств	**Сотрудник** - адрес - город - улица - дом

Бинарные связи обозначаются линиями с указанием имени связи и мощности связи со стороны каждой сущности. Мощность связи определяет количество объектов, соединяемых с каждым объектом на другом конце связи:

- **1..*** — один или более;
- **0..*** — ноль или более;
- **1** — ровно один;
- **0..1** — может быть один.

Может быть задан также диапазон (например, 1..10) или точное количество, отличное от 1.

Для обозначения *n*-арной связи используется ромб, внутри которого указывается имя связи. Кратность *n*-арной связи для каждой сущности показывает, каким может быть количество ее объектов в случае, если со стороны остальных сущностей-участниц в связи участвует ровно по одному объекту. Например, трехсторонняя связь на рис. 5.6 определяет, что поставлять партии любых деталей для любого проекта может любой поставщик.

Рис. 5.6. Пример трехсторонней связи

Присутствие в задании мощности связи значения «0» объявляет связь как неполную (необязательную). Например, связь РАБОТАЕТ между сущностями ОТДЕЛ и СОТРУДНИК на рис. 5.5 должна читаться следующим образом: «Каждый сотрудник обязательно работает только в одном отделе; в отделе работает произвольное число сотрудников или не работает ни одного».

Связь может быть модифицирована указанием роли. Например, для рекурсивной связи СОСТАВ на рис. 5.5 указаны роли: «Деталь *состоит из* ...» и «Деталь *в составе*...».

Связь «многие ко многим» может иметь собственные свойства, характеризующие взаимодействие сущностей (например, связи

УЧАСТИЕ и РЕАЛИЗАЦИЯ ПРОЕКТА на рис. 5.5). В этом случае для изображения связи используется графический элемент, соответствующий слабой сущности. Прямоугольник сущности присоединяется к линии связи (или к ромбу в случае *n*-арной связи) пунктирной линией.

Нотация Чена

Каждый тип сущности в нотации Чена представляется в виде прямоугольника, содержащего имя сущности (существительное в единственном числе).

Связь (и бинарная, и *n*-арная) на ER-диаграмме отображается в виде ромба с именем связи внутри. В качестве имени обычно используются отглагольные существительные.

Свойства отображаются в виде эллипсов, содержащих имя свойства. Эллипс соединяется с соответствующей сущностью или связью линией (рис. 5.7).

Рис. 5.7. Фрагмент ER-диаграммы в нотации Чена

Участники связи соединены со связью линиями. Двойная линия обозначает полное (обязательное) участие сущности в связи с данной стороны. Например, обязательная связь РУКОВОДСТВО со стороны сущности ПРОЕКТ (рис. 5.8) означает, что у каждого проекта должен быть руководитель. Однако не каждый сотрудник должен руководить проектом.

Рис. 5.8. Пример полной и неполной связей

Тип связи указывается индексами «1» или «M» над соответствующей линией. Например, связь РУКОВОДСТВО (рис. 5.8) имеет тип «один ко многим»: один сотрудник может руководить многими про-

ектами; связь УЧАСТИЕ (рис. 5.7) имеет тип «многие ко многим»: один сотрудник может участвовать во многих проектах, и в проекте могут участвовать многие сотрудники.

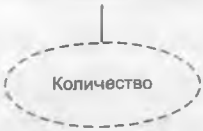
Для отражения слабых (зависимых) сущностей используются прямоугольники, стороны которых рисуются двойными линиями. В представленном на рис. 5.9 фрагменте ER-диаграммы ПОДЧИНЕННЫЙ — сущность слабого типа.

Рис. 5.9. Пример слабой сущности

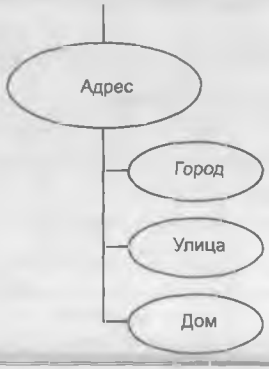
Стороны ромба рисуют двойными линиями, если это связь сущности слабого типа с сущностью, от которой она зависит. Например, связь «Подчинение», связывающая сущность слабого типа ПОДЧИНЕННЫЙ с сущностью СОТРУДНИК, от которой она зависит.

Правила изображения свойств различных типов в нотации Чена приведены в табл. 5.3.

Таблица 5.3. **Изображение типов свойств в нотации Чена**

Тип свойства	Описание	Пример
Свойство первичного ключа	Имя свойства подчеркивается	
Многозначное	Контур эллипса рисуется двойной линией	
Производное	Контур эллипса рисуется штриховой линией	
Условное (необязательное)	Эллипс соединяется с прямоугольником сущности пунктирной линией	

Окончание табл. 5.3

Тип свойства	Описание	Пример
Составное	Составляющие свойства отображаются другими эллипсами, соединенными с эллипсом составного	

5.4. Функциональная модель IDEF0

Модель IDEF0 является частью семейства стандартов IDEF и представляет собой описание системы в целом как множества взаимозависимых действий или функций, причем IDEF0-функции системы исследуются независимо от объектов, которые обеспечивают их выполнение.

Наиболее часто модель IDEF0 используется при проектировании систем на концептуальной стадии разработки, для сбора данных и моделирования процессов «как есть» («as is»). При построении модели необходимо определить:

1) назначение модели — набор вопросов, на которые должна ответить модель;

2) границы моделирования — ширину охвата предметной области и глубину детализации;

3) целевую аудиторию, для нужд которой создается модель;

4) точку зрения, с которой наблюдается система при построении модели. Точка зрения должна учитывать обозначенные назначение модели и границы моделирования и должна быть неизменной для всех элементов модели.

Главной организационной единицей модели является диаграмма. Графический язык модели содержит всего два элемента — блоки (функции) и стрелки (связи).

IDEF0-модель представляет собой иерархическое множество вложенных последовательностей функциональных блоков, поэтому в первую очередь должна быть определена функция, описывающая систему в целом, — *контекстная функция*. Далее в процессе построения модели любой блок может быть декомпозирован на составляющие его блоки.

Блок на функциональной диаграмме изображается именованным прямоугольником (рис. 5.10). В полное описание блока входит четыре типа стрелок, каждый из которых соединяется с определенной стороной функционального блока:

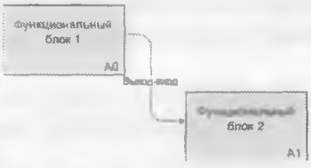
- I (Input) — вход — потребляемые и/или преобразуемые данные;
- C (Control) — управление — ограничения и инструкции, влияющие на ход выполнения процесса;
- O (Output) — выход — данные, получаемые в результате работы функции;
- M (Mechanism) — механизм, который используется для выполнения процесса, но остается неизменным.

Рис. 5.10. Функциональный блок диаграммы IDEF0

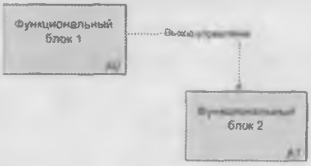
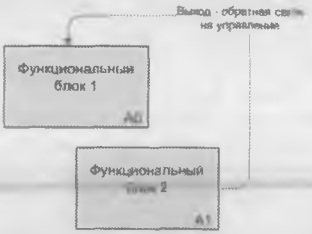
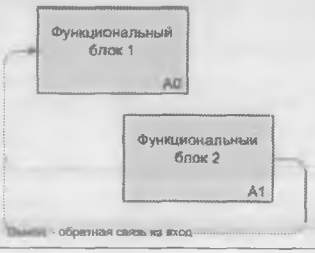
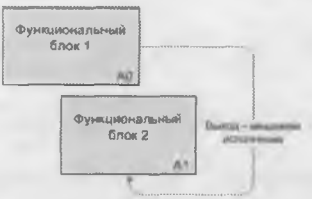
Каждый функциональный блок должен иметь как минимум по одной стрелке входа, выхода и управления.

Комбинированные *стрелки* соединяют функциональные блоки и определяют порядок выполнения функций, передачи информации и управления. В табл. 5.4 представлены пять основных видов соединений.

Таблица 5.4. **Виды комбинированных стрелок**

Графика	Название	Назначение
	«Выход — вход»	Одна из функций должна полностью завершиться перед началом другой (выходная информация одной функции служит входом для другой)

Окончание табл. 5.4

Графика	Название	Назначение
	«Выход — управление»	Выход одного блока управляет работой другого
	«Выход — обратная связь на управление»	Зависимый блок формирует обратную связь на управление
	«Выход — обратная связь на вход»	Описание циклов повторной обработки
	«Выход — механизм исполнения»	Выход одного блока является инструментом для исполнения другого

5.5. Метод моделирования IDEF3

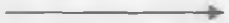
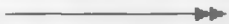
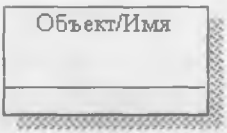
Метод является частью семейства стандартов IDEF и предназначен для построения моделей таких процессов, в которых важно понять последовательность выполнения действий и взаимозависимости между ними. Метод IDEF3 может служить дополнением к методу функционального моделирования IDEF0 (для детализации функциональных блоков IDEF0, не имеющих диаграмм декомпозиции). Ос-

новой модели IDEF3 служит так называемый сценарий процесса, который выделяет последовательность действий и подпроцессов анализируемой системы.

Графический язык модели содержит следующие элементы (табл. 5.5):

- действия;
- связи;
- соединения;
- указатели.

Таблица 5.5. **Графический язык модели IDEF3**

Графика	Название	Назначение
	Действие	Моделирует некоторое действие, преобразующее вход в выход. По своему назначению почти идентично функциональным блокам IDEF0 и DFD
	Связь типа «Временное предшествование»	Обозначает, что исходное действие должно завершиться прежде, чем начнется конечное действие
	Связь типа «Объектный поток»	Ообозначает, что выход исходного действия является входом конечного действия (очевидно, включает и связь «Временное предшествование»)
	Связь типа «Нечеткое отношение»	Используется, когда отношение между действиями не соответствует ни одному из предыдущих типов; значение каждой такой связи должно определяться аналитиком отдельно
	Указатель	Специальный элемент, позволяющий комментировать и пояснять другие элементы модели
	И-соединение	Инициирует выполнение конечных действий, т. е. все действия, входящие в И-соединение, должны быть завершены перед выполнением исходящих из него действий

Окончание табл. 5.5

Графика	Название	Назначение
	Эксклюзивное ИЛИ-соединение	Инициирует только одно из исходящих действий (для разворачивающего соединения) после того, как только одно из входящих действий (для сворачивающего соединения) будет выполнено
	ИЛИ-соединение	Предназначено для ситуаций, которые не могут быть описаны двумя предыдущими типами соединений, т. е., может быть, несколько действий должно закончиться (для сворачивающего соединения), прежде чем будет инициировано одно или несколько действий (для разворачивающего соединения)

Действия изображаются в виде прямоугольника, содержащего имя действия и составной номер действия, который включает номер родительского действия (на рисунке — 1) и непосредственный номер самого действия (на рисунке — 5), разделенные точкой.

Связи определяют взаимоотношения между действиями, являются однонаправленными и изображаются стрелками, вид которых соответствует типу связи.

Указатели — специальные элементы, которые ссылаются на другие элементы модели (для привлечения внимания к важным аспектам модели). Указатель изображается в виде прямоугольника, похожего на действие. Имя указателя обязательно включает его тип.

Допустимы следующие типы указателей:

- *Объект* (Object) — для описания объекта, принимающего участие в действии;
- *Ссылка* (GoTo) — для реализации цикличности выполнения действий;
- *Единица действия* (Unit of Behavior —UOB) — для многократного изображения на диаграмме одного и того же действия;
- *Заметка* (Note) — для документирования информации общего характера;
- *Уточнение* (Elaboration — Elab) — для уточнения или более подробного описания элемента, изображенного на диаграммс.

Соединения используются для описания ветвления процесса. *Разворачивающее соединение* описывает процесс, когда завершение одного действия инициирует начало выполнения сразу нескольких действий. *Сворачивающее соединение* применяется, когда некоторое действие требует предварительного завершения нескольких предшествующих. В табл. 5.5 приведены возможные типы соединений.

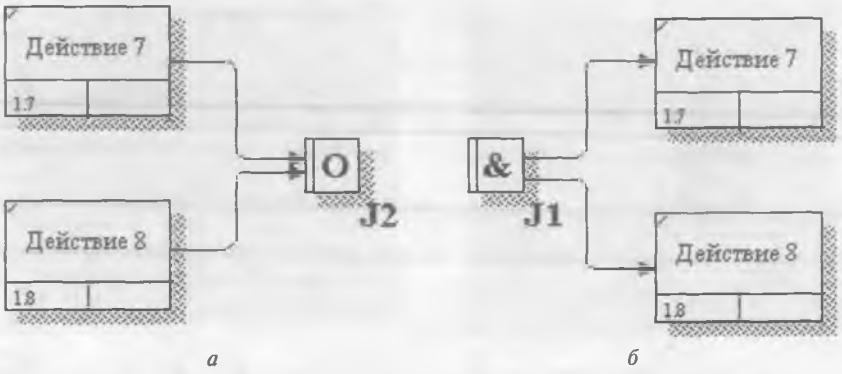


Рис. 5.11. Примеры соединений:
а — сворачивающее; *б* — разворачивающее

На рис. 5.11 приведены примеры сворачивающего ИЛИ-соединения и разворачивающего И-соединения.

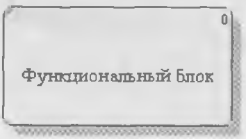
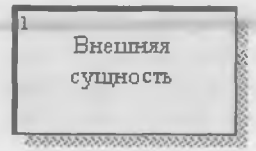
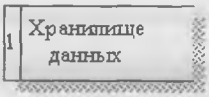
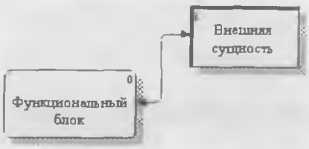
5.6. Диаграммы потоков данных

Диаграммы потоков данных (Data Flow Diagrams — DFD) моделируют систему как набор функциональных процессов, связанных потоками данных. Цель такого представления — продемонстрировать, как каждый процесс преобразует свои входные данные в выходные, а также выявить отношения между этими процессами.

Диаграммы потоков данных содержат (в отличие от функциональной диаграммы IDEF0) два новых типа объектов — хранилища данных и внешние сущности. Эти объекты позволяют определять взаимодействие с частями системы (или другими системами), которые выходят за границы моделирования. Стрелки в DFD отражают такие характеристики системы, как движение объектов (потоки данных), хранение объектов (хранилища данных), источники и потребители данных (внешние сущности).

Для построения DFD традиционно используются две различные нотации, соответствующие методам Йордона — Де Марко и Гейна — Сэрсона. Эти нотации незначительно отличаются друг от друга графическим изображением символов. В табл. 5.6 приведен графический язык нотации Гейна — Сэрсона.

Таблица 5.6. **Графический язык модели DFD (нотация Гейна — Сэрсона)**

Графика	Название	Назначение
	Функциональный блок	Моделирует некоторую функцию, преобразующую вход в выход. По своему назначению почти идентичен функциональным блокам IDEF0 и действиям IDEF3
	Внешняя сущность	Обеспечивают необходимые входы и (или) выходы для функциональных блоков. Одна и та же внешняя сущность может функционировать и как поставщик, и как получатель данных и может повторяться на диаграмме несколько раз
	Хранилище данных	Механизм, который поддерживает хранение данных для их промежуточной обработки
	Потоки данных	Описывают перемещение данных между частями системы. Стрелки, обозначающие потоки, могут быть как однонаправленными, так и двунаправленными, а также могут начинаться и заканчиваться на любой стороне блока

Потоки данных на диаграмме могут представлять ветвление и объединение данных. *Ветвление* обозначает декомпозицию данных, перемещаемых потоком. При этом все полученные после ветвления подпотоки должны быть поименованы (рис. 5.12, *а*). *Объединение* обозначает соединение данных для формирования комплексных объектов данных (рис. 5.12, *б*). Так же как и при ветвлении, все части комплексного объекта и сам получаемый объект данных должны быть поименованы.

Рис. 5.12. Потоки данных:
а — объединение; *б* — ветвление

Контекстная диаграмма DFD (т. е. диаграмма, описывающая систему в целом) обычно состоит из одного функционального блока и нескольких внешних сущностей, с которыми система обменивается данными. Имя функционального блока на такой диаграмме совпадает с именем системы.

Контрольные вопросы

1. Охарактеризуйте графический язык диаграммы потоков данных (DFD).
2. Охарактеризуйте графический язык функциональной модели IDEF0.
3. Охарактеризуйте графический язык модели IDEF3.
4. Определите типологию связей.
5. Определите соотношение понятий «сущность» и «связь».
6. Какими свойствами должен обладать индентификатор объекта?
7. Проанализируйте соотношение ER-модели и ER-диаграммы.

Глава 6
ЛОГИЧЕСКИЕ МОДЕЛИ БД

Модель логического уровня проектирования БД — модель данных — представляет собой сочетание трех компонентов.

1. Структурный компонент, т. е. набор правил, по которым может быть построена БД.

2. Управляющий компонент, определяющий типы допустимых операций с данными (сюда относятся операции обновления и извлечения данных, а также операции изменения структуры БД).

3. Поддержка (необязательная) набора ограничений целостности данных, гарантирующая корректность используемых данных.

6.1. Модели на основе записей

Структурный компонент модели рассматривают с точки зрения записей. В такой модели структуру данных составляет совокупность нескольких типов записей, каждая из которых имеет фиксированный формат: 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);
- иерархическая модель данных (hierarchical data model).

Реляционная модель данных (см. главу 2) основана на понятии *математических отношений*. Для работы с БД не требуются знания физической организации данных, связи реализуются аппаратом внешних ключей, подход к обработке данных — декларативный.

На рис. 6.1 показан пример реляционной схемы из двух отношений, содержащих сведения о кафедрах вуза и кадровом составе. Например, из таблицы «Кадровый состав» видно, что сотрудник Ива-

Структура

Каф	*Телефон*	*Корпус*	*№ ком*
22	25-15	А	322
23	36-42	В	221
24	99-18	Б	117

Кадровый состав

Таб№	*ФИО*	*Должность*	*Каф*
121	Иванов И.И.	Зав. каф.	22
231	Сидоров С.С.	Проф.	22
123	Гнацинтова Г.Г.	Проф.	23
432	Цветкова С.С.	Доцент	23
465	Козлов К.К.	Доцент	24
675	Петров П.П.	Ст.преп.	24
782	Лютикова Л.Л.	Ассистент	22

Рис. 6.1. Реляционная схема на основе двух отношений

нов И.И. работает в должности заведующего кафедрой 22, которая, согласно данным из таблицы «Структура», расположена в корпусе А, в комнате 322. Связь «Сотрудник *работает* на кафедре» между отношениями «Кадровый состав» и «Структура» явно не задана, а существует на уровне семантически одинаковых атрибутов: атрибута *Каф* в отношении «Кадровый состав» и атрибута *Каф* в отношении «Структура».

В реляционной модели данных база данных с точки зрения пользователя выглядит как набор таблиц (отношений). Однако следует отметить, что такое восприятие относится только к логической структуре базы данных, т. е. к внешнему и концептуальному уровням архитектуры ANSI/SPARC и не определяет физическую структуру базы данных, которая может быть реализована с помощью различных способов хранения.

Иерархическая и сетевая модели данных были созданы значительно раньше реляционной модели, потому их связь с концепциями традиционной обработки файлов более очевидна. Для работы с БД требуются знания физической организации данных, связи реализуются средствами ссылочного аппарата, подход к обработке данных — навигационный.

В *сетевой модели* данные представлены в виде коллекций *записей*, а связи — в виде *наборов*, реализованных с помощью указателей

Рис. 6.2. Фрагмент сетевой схемы

(рис. 6.2). Сетевая модель представляется графом, в котором узлами являются записи, а ребрами — наборы.

Как пример сетевой СУБД можно выделить систему IDMS/R фирмы Computer Associates.

Иерархическая модель является ограниченным подтипом сетевой модели, представляемой древовидным графом (рис. 6.3), т. е. в иерархической модели узел может иметь только одного родителя. Связи типа «многие ко многим» в рамках иерархической модели могут быть реализованы только с дублированием данных.

Примером в свое время распространенной иерархической СУБД служит система IMS корпорации IBM.

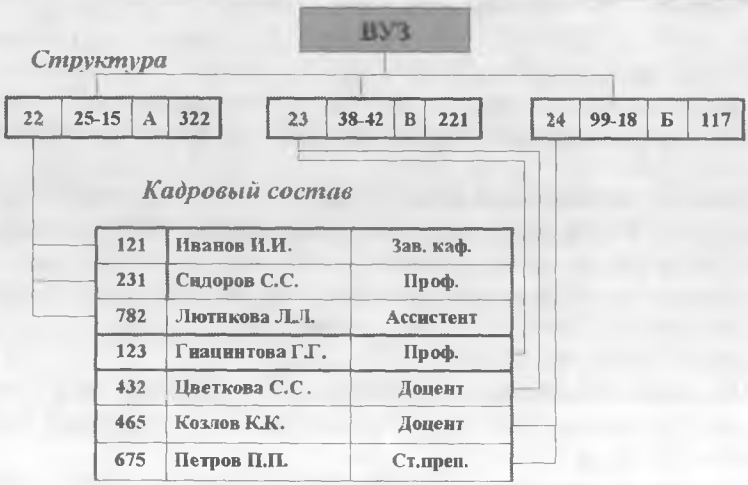


Рис. 6.3. Фрагмент иерархической схемы

6.2. Реляционная модель данных

Основные понятия реляционной модели изложены в главе 2. Структурный компонент реляционной модели — отношение (relation, отсюда и произошло название модели — реляционная). Множество допустимых операций над данными, представленными в виде совокупности отношений, задается реляционной алгеброй. Кроме реляционных операций манипулирования данными, в состав управляющего компонента должны входить средства определения данных, условий целостности и представлений.

6.2.1. Целостность данных

Целостность данных в реляционной модели рассматривается на трех уровнях.

Целостность на уровне доменов. Домен в реляционной модели представляет собой множество значений *простого* (т. е. не обладающего внутренней структурой) типа данных, и все значения атрибутов отношения считаются атомарными.

Домен имеет уникальное имя в пределах базы данных. С логической точки зрения некорректно сравнивать значения из семантически различных доменов, даже если они имеют одинаковый тип.

Целостность на уровне отношений. Средством адресации на уровне кортежей в реляционной модели служат потенциальные (уникально характеризующие кортеж) ключи.

Потенциальные ключи идентифицируют объекты предметной области — экземпляры сущностей, данные о которых хранятся в отношениях. Поскольку эти экземпляры должны быть различимы по определению, их идентификаторы не могут содержать неизвестные значения (или NULL-значения). Этот факт позволяет сформулировать правило целостности отношений: в каждом отношении должен быть по крайней мере один потенциальный ключ, атрибуты которого не содержат NULL-значений. Этот потенциальный ключ лучше всего объявлять первичным ключом отношения.

Целостность внешних ключей (целостность на уровне БД). Для связи объектов предметной области в реляционной модели используется аппарат внешних ключей.

Внешние ключи должны быть согласованными, т. е. для каждого отдельного значения внешнего ключа в дочернем отношении должно

существовать соответствующее значение в поле связи в родительском отношении. Это правило называется *правилом целостности внешних ключей* или *ссылочной целостности* реляционной базы данных: При этом в поле связи родительской таблицы могут присутствовать значения, на которые не ссылается ни одно из значений внешнего ключа. Для атрибутов внешнего ключа допустимы также и NULL-значения

6.2.2. Правила Кодда

В целом концепция реляционной модели определяется следующими двенадцатью правилами Кодда (приводятся по [4]).

1. *Правило информации.* Вся информация в базе данных должна быть предоставлена исключительно на логическом уровне и только одним способом — в виде значений, содержащихся в таблицах.

2. *Правило гарантированного доступа.* Логический доступ ко всем и каждому элементу данных (атомарному значению) в реляционной базе данных должен обеспечиваться путем использования комбинации имени таблицы, первичного ключа и имени столбца.

Правило указывает на роль первичных ключей при поиске информации в базе данных. Имя таблицы позволяет найти требуемую таблицу, имя столбца позволяет найти требуемый столбец, а первичный ключ позволяет найти строку, содержащую искомый элемент данных.

3. *Правило поддержки недействительных значений.* В реляционной базе данных должна быть реализована поддержка недействительных значений, которые отличаются от строки символов нулевой длины, строки пробельных символов, от нуля или любого другого числа и используются для представления отсутствующих данных независимо от типа этих данных.

Правило требует, чтобы отсутствующие данные можно было представить с помощью недействительных значений (null).

4. *Правило динамического каталога, основанного на реляционной модели.* Описание базы данных на логическом уровне должно быть представлено в том же виде, что и основные данные, чтобы пользователи, обладающие соответствующими правами, могли работать с ним с помощью того же реляционного языка, который они применяют для работы с основными данными.

Правило говорит о том, что реляционная база данных должна сама себя описывать. Другими словами, база данных должна содер-

жать набор *системных таблиц,* описывающих структуру самой базы данных.

5. *Правило исчерпывающего подъязыка данных.* Реляционная система может поддерживать различные языки и режимы взаимодействия с пользователем (например, режим вопросов и ответов). Однако должен существовать по крайней мере один язык, операторы которого можно представить в виде строк символов в соответствии с некоторым четко определенным синтаксисом и который в полной мере поддерживает следующие элементы:

- определение данных;
- определение представлений;
- обработку данных (интерактивную и программную);
- условия целостности;
- идентификацию прав доступа;
- границы транзакций (начало, завершение и отмена).

То есть правило требует, чтобы СУБД использовала язык реляционной базы данных, например SQL. Такой язык должен поддерживать все основные функции СУБД — создание базы данных, чтение и ввод данных, реализацию защиты базы данных и т. д.

6. *Правило обновления представлений.* Все представления, которые теоретически можно обновить, должны быть доступны для обновления.

Правило касается *представлений,* которые являются виртуальными таблицами, позволяющими показывать различным пользователям различные фрагменты структуры базы данных. Это одно из правил, которые сложнее всего реализовать на практике

7. *Правило добавления, обновления и удаления.* Возможность работать с отношением как с одним операндом должна существовать не только при чтении данных, но и при добавлении, обновлении и удалении данных.

Правило акцентирует внимание на том, что базы данных по своей природе ориентированы на множества. Оно требует, чтобы операции добавления, удаления и обновления можно было выполнять над множествами строк. Это правило предназначено для того, чтобы запретить реализации, в которых поддерживаются только операции над одной строкой.

8. *Правило независимости физических данных.* Прикладные программы и утилиты для работы с данными должны на логическом уровне оставаться нетронутыми при любых изменениях способов хранения данных или методов доступа к ним.

9. *Правило независимости логических данных.* Прикладные программы и утилиты для работы с данными должны на логическом уровне оставаться неизменными при внесении в базовые таблицы любых изменений, которые теоретически позволяют сохранить нетронутыми содержащиеся в этих таблицах данные.

Это и восьмое правила означают отделение пользователя и прикладной программы от низкоуровневой реализации базы данных. Они утверждают, что конкретные способы реализации хранения или доступа, используемые в СУБД, и даже изменения структуры таблиц базы данных не должны влиять на возможность пользователя работать с данными.

10. *Правило независимости условий целостности.* Должна существовать возможность определять условия целостности, специфические для конкретной реляционной базы данных, на подъязыке реляционной базы данных и хранить их в каталоге, а не в прикладной программе.

Правило требует, чтобы язык базы данных поддерживал ограничительные условия, налагаемые на вводимые данные и действия, которые могут быть выполнены над данными.

11. *Правило независимости распространения.* Реляционная СУБД не должна зависеть от потребностей конкретного клиента.

Правило говорит о том, что язык базы данных должен обеспечивать возможность работы с распределенными данными, расположенными на других компьютерных системах.

12. *Правило единственности.* Если в реляционной системе есть низкоуровневый язык (обрабатывающий одну запись за один раз), то должна отсутствовать возможность использования его для того, чтобы обойти правила и условия целостности, выраженные на реляционном языке высокого уровня (обрабатывающем несколько записей за один раз).

Правило предотвращает использование других возможностей для работы с базой данных, помимо языка базы данных, поскольку это может нарушить ее целостность.

6.2.3. Нормализация отношений

При работе с отношениями, содержащими избыточные данные, могут возникать проблемы, которые называются *аномалиями обновления* и подразделяются на аномалии вставки, аномалии удаления и

аномалии модификации. Рассмотрим, например, отношение, представленное в виде таблицы на рис. 6.4.

Проект	Руководитель проекта	Должность руководителя	Исполнитель проекта	Должность исполнителя	Стоимость работы исполнителя по проекту
П12.1	Самойлов Ю.В.	Зав. отделом	Иванов И.И.	Конструктор	30 000
П12.1	Самойлов Ю.В.	Зав. отделом	Мальцев К.Н.	Инженер	25 000
П12.1	Самойлов Ю.В.	Зав. отделом	Петров А.К.	Техник	10 000

Рис. 6.4. Пример отношения

Аномалии вставки. В реляционную таблицу нельзя добавить, например, информацию о новом сотруднике, если он еще не участвует ни в одном проекте. С другой стороны, добавление нового сотрудника как участника одного из проектов потребует обязательного дублирования сведений о руководителе проекта, что ведет к потенциальной несовместимости данных (в случае ошибок при вводе).

Аномалии удаления. При удалении из реляционной таблицы информации о сотрудниках, работающих над конкретным проектом (например, при смене команды), будет полностью удалена информация о самом проекте.

Аномалии модификации. Вызывают потенциальную противоречивость данных, которая возникает при вводе повторяющихся данных (в случае ошибочного ввода в одно или несколько значений), а также при редактировании повторяющихся данных.

Перечисленных аномалий можно избежать путем нормализации исходного отношения.

Процесс нормализации — это декомпозиция таблицы на две или более с целью ликвидации дублирования данных и потенциальной их противоречивости. Окончательная цель нормализации сводится к получению такого проекта базы данных, в котором «каждый факт появляется лишь в одном месте».

В основе процесса нормализации лежит концепция функциональной зависимости.

Функциональная зависимость описывает связь между атрибутами отношения: если в отношении R, содержащем атрибуты A и B, атрибут B функционально зависит от атрибута A, то каждое отдельное значение атрибута A связано только с одним значением атрибута B (причем в качестве A и B могут выступать группы атрибутов). Атрибут или

группа атрибутов *A* называются при этом *детерминантом* функциональной зависимости.

Таким образом, при наличии функциональной зависимости $A \rightarrow B$ кортежи (строки), имеющие одинаковое значение атрибута *A*, совпадают и по значению атрибута *B*. Однако обратное неверно: одно и то же значение атрибута *B* может соответствовать разным значениям атрибута *A*. Например, из функциональной зависимости *Исполнитель проекта* → *Должность исполнителя* (см. рис. 6.4) следует, что везде, где будет указываться сотрудник «Иванов И.И.», ему будет соответствовать должность «конструктор», но должность «конструктор» могут иметь и другие сотрудники.

В таблице на рис. 6.4 между атрибутами существуют следующие функциональные зависимости:

1. *Проект* → *Руководитель проекта*.
2. *Руководитель проекта* → *Должность руководителя*.
3. *Исполнитель проекта* → *Должность исполнителя*.
4. *Проект*, *Исполнитель проекта* → *Стоимость работы исполнителя по проекту*.

Функциональная зависимость $A \rightarrow B$ является *полной* функциональной зависимостью, если удаление какого-либо атрибута из группы атрибутов *A* приводит к потере этой зависимости. Зависимость под номером четыре служит примером полной функциональной зависимости.

Функциональная зависимость $A \rightarrow B$ является *частичной* функциональной зависимостью, если в группе атрибутов *A* есть один или несколько атрибутов, при удалении которых эта зависимость сохраняется. Например, функциональная зависимость *Проект*, *Исполнитель проекта* → *Должность исполнителя* сохранится при удалении из детерминанта атрибута *Проект*.

Если для атрибутов *A*, *B* и *C* некоторого отношения существуют функциональные зависимости $A \rightarrow B$, $B \rightarrow C$, говорят, что атрибут *C* связан *транзитивной зависимостью* с атрибутом *A* через атрибут *B* (при этом атрибут *A* не должен функционально зависеть ни от атрибута *B*, ни от атрибута *C*). Например, функциональные зависимости 1 и 2 порождают транзитивную зависимость атрибута *Должность руководителя* от атрибута *Проект* через атрибут *Руководитель проекта*.

Многозначная зависимость. Говорят, что один атрибут отношения многозначно определяет другой атрибут того же отношения, если для каждого значения первого атрибута существует определенное множество соответствующих значений второго атрибута (т. е. между атрибу-

Проект	Оборудование
П12.1	Сканер
П12.1	Ксерокс

Рис. 6.5. Тривиальная многозначная зависимость

тами задана связь «один ко многим»). Например, в таблице, изображенной на рис. 6.5, определена многозначная зависимость между проектом и оборудованием для его реализации. Такая многозначная зависимость называется *тривиальной* и не накладывает ограничений на ввод данных в таблицу.

Если же в отношении существуют многозначные зависимости между атрибутами *A* и *B*, *A* и *C*, при которых атрибуты *B* и *C* не зависят друг от друга, то такая многозначная зависимость называется *нетривиальной*. В качестве примера рассмотрим фрагмент таблицы «Поставка оборудования для выполнения проектов» (рис. 6.6). Таблица отражает связь проектов с поставщиками и оборудованием. В этой таблице существует многозначная зависимость *Проект — Поставщик*: для проекта П12.1 поставку оборудования осуществляют поставщики «Орфей» и «Атлант» и, соответственно, каждый из них может поставить сканеры и ксероксы. Другая многозначная зависимость — *Проект — Оборудование*: для проекта П12.1 необходимо поставить сканер и ксерокс. При этом атрибуты *Оборудование* и *Поставщик* не связаны функциональной зависимостью, что приводит к появлению избыточности (чтобы добавить новое оборудование, придется ввести в таблицу две новых строки).

Проект	Оборудование	Поставщик
П12.1	Сканер	«Орфей»
П12.1	Сканер	«Атлант»
П12.1	Ксерокс	«Орфей»
П12.1	Ксерокс	«Атлант»

Рис. 6.6. Нетривиальная многозначная зависимость

Проект	Оборудование
П12.1	Сканер
П12.1	Ксерокс

Проект	Поставщик
П12.1	«Орфей»
П12.1	«Атлант»

Рис. 6.7. Декомпозиция отношения рис. 6.6

6.2.4. Нормальные формы отношений

На каждом этапе нормализации каждое из отношений находится в одной из так называемых *нормальных форм*. Нормальные формы (от самой младшей до самой старшей) связаны операцией включения, т. е. более старшая нормальная форма обладает свойствами всех предшествующих и дополнительно имеет свои отличительные признаки.

Нормализация представляет собой формальный метод анализа отношений на основе выявления первичного ключа и существующих функциональных зависимостей. Последовательное удаление частичных функциональных зависимостей и транзитивных зависимостей осуществляется путем декомпозиции отношений и перевода их в следующую (более старшую) нормальную форму.

Отношение находится в первой нормальной форме (*1НФ*), если:

- каждое значение любого его атрибута является атомарным;
- в отношении отсутствуют одинаковые кортежи;
- каждый атрибут уникально поименован и содержит атомарное значение внутри каждого кортежа;
- каждый атрибут ассоциирован с определенным доменом (типом данных).

Отношение, находящееся в 1НФ, имеет *первичный ключ* — атрибут или совокупность атрибутов, значения которых уникально характеризуют каждый кортеж.

Первичным ключом отношения на рис. 6.4 является совокупность атрибутов *Проект*, *Исполнитель проекта*.

Отношение находится во второй нормальной форме (*2НФ*), если оно удовлетворяет определению 1НФ и все его атрибуты, не входящие в первичный ключ, связаны полной функциональной зависимостью с первичным ключом. Отношение на рис. 6.4 не находится в 2НФ, так как атрибуты *Руководитель проекта* и *Исполнитель проекта* частично зависят от первичного ключа.

Отношение находится в третьей нормальной форме (*3НФ*), если оно удовлетворяет определению 2НФ и ни один из его не ключевых атрибутов не связан транзитивной функциональной зависимостью с первичным ключом (т. е. ни один из не ключевых атрибутов не связан функциональной зависимостью с любым другим не ключевым атрибутом). Рассмотрим, например, отношение, состоящее из атрибутов *Проект*, *Руководитель проекта* и *Должность руководителя*. Первичным ключом такого отношения является атрибут *Проект*, и выполняются требования 2НФ. Однако отношение не находится в 3НФ, так

как атрибут *Должность руководителя* транзитивно зависит от атрибута *Проект* через атрибут *Руководитель проекта*.

Отношение находится в *третьей нормальной форме Бойса —Кодда (НФБК)* (усиленная третья нормальная форма) тогда и только тогда, когда любая функциональная зависимость между его атрибутами сводится к полной функциональной зависимости от *возможного* первичного ключа (т. е. все детерминанты отношения являются потенциальными первичными ключами).

Рассмотрим пример отношения, описывающего результаты проверки выполнения отдельных работ по проектам:

```
Результат проверки (Проект, Дата, Время,
    Наименование работы,Выполненный объем, Проверяющий)
```

Пусть особенности предметной области таковы, что в день отдельный Проект проверяется только один раз, и каждый Проверяющий фиксирует в один день объем выполнения работы только одного наименования. Тогда в отношении можно выделить следующие потенциальные ключи:

Проект, Дата

Дата, Время, Проверяющий

Относительно первичного ключа *Проект, Дата* отношение находится в 3НФ, однако оно содержит функциональную зависимость *Дата, Проверяющий* → *Наименование работы*, детерминант которой не является потенциальным ключом, т. е. отношение не соответствует требованиям НФБК.

В следующих нормальных формах (4НФ и 5НФ) учитываются не только функциональные, но и многозначные зависимости между атрибутами.

Отношение находится в *четвертой нормальной форме (4НФ)*, если оно находится в НФБК и не содержит нетривиальных многозначных зависимостей. Отношение на рис. 6.5 отвечает требованиям 4НФ, а отношение на рис. 6.6 не находится в 4НФ. Для преобразования к 4НФ проводят декомпозицию отношения на две проекции (в соответствии с тривиальными многозначными зависимостями). Декомпозиция отношения рис. 6.6 представлена на рис. 6.7.

Предположим теперь, что при заполнении таблицы на рис. 6.6 необходимо было применять следующее правило: если поставщик поставляет какое-либо оборудование для реализации проекта, то он должен поставить это же оборудование для реализации всех других проектов, для которых осуществлял какие-либо поставки (последняя

строка в таблице на рис. 6.8 появилась именно в соответствии с этим правилом). В этом случае в естественном соединении двух проекций (таких, как, например, на рис. 6.7) получатся лишние строки. Таким образом, имеется ситуация *зависимости соединения*. Требования 4НФ при этом выполняются, так как атрибуты *Оборудование* и *Поставщик* связаны между собой.

Пятая нормальная форма (5НФ) требует отсутствия в отношении зависимостей соединения. Для приведения к 5НФ необходимо включить в декомпозицию еще одно отношение, отражающее зависимость соединения. На рис. 6.9 показана декомпозиция отношения рис. 6.8. Следует отметить, что никакие из двух отношений декомпозиции не дают в естественном соединении исходное: для его восстановления необходимо соединить все три проекции.

Проект	Оборудование	Поставщик
П12.1	Сканер	«Атлант»
П12.1	Ксерокс	«Орфей»
П33.3	Сканер	«Орфей»
П12.1	Сканер	«Орфей»

Рис. 6.8. Отношение с зависимостью соединения

Проект	Оборудование
П12.1	Сканер
П12.1	Ксерокс
П33.3	Сканер

Проект	Поставщик
П12.1	«Орфей»
П12.1	«Атлант»
П33.3	«Орфей»

Оборудование	Поставщик
Сканер	«Орфей»
Ксерокс	«Атлант»
Сканер	«Орфей»

Рис. 6.9. Декомпозиция отношения рис. 6.8

Следует отметить, что на практике обычно довольствуются приведением реляционной БД к 3НФ или к НФБК.

6.2.5. Процедура нормализации

Процедура приведения отношений к 3НФ основывается на том, что единственными функциональными зависимостями в любой таблице должны быть зависимости вида $A \rightarrow K$, где K — первичный ключ, а A — некоторый атрибут. Цель нормализации состоит в удалении других функциональных зависимостей.

Возможны два случая.

1. Отношение имеет составной первичный ключ, например (*K1*, *K2*), и включает также атрибут *A*, который функционально зависит от части этого ключа (например, от *K2*), но не от полного ключа. В этом случае рекомендуется сформировать другое отношение, содержащее атрибуты *K2* и *A* (первичный ключ — *K2*), и удалить атрибут *A* из исходного отношения.

2. Отношение имеет первичный ключ *K*, атрибут *A1*, который не является возможным ключом, но функционально зависит от *K*, и другой не ключевой атрибут *A2*, который функционально зависит от *A1* (случай транзитивной зависимости). Решение здесь, по существу, то же самое, что и прежде, — формируется другое отношение, содержащее атрибуты *A1* и *A2*, с первичным ключом *A1*, а атрибут *A2* удаляется из исходного отношения.

Таким образом, повторяя применение двух рассмотренных правил, для любого отношения почти во всех реальных практических ситуациях можно получить в конечном счете множество отношений, которые находятся в 3НФ и не содержат каких-либо функциональных зависимостей вида, отличного от $A \rightarrow K$.

Для приведения отношений к НФБК необходимо применить процедуру нормализации ко всем потенциальным ключам.

6.2.6. Получение реляционной схемы из ER-диаграммы

Как было отмечено в п. 5.3, формальный язык ER-диаграммы дает возможность сформулировать конечный набор правил проектирования модели логического уровня. Основные правила получения реляционной схемы из ER-диаграммы перечислены ниже.

1. Каждая простая сущность превращается в таблицу (отношение). Имя сущности становится именем таблицы.

2. Связь «многие ко многим» рассматривается как сущность—связь и превращается в таблицу (отношение). Тем самым связь «многие ко многим» трансформируется в две связи «многое к одному». Такое отношение отражает динамику предметной области и рассматривается как зависимый объект модели, поэтому связи «многие к одному» носят идентифицирующий характер.

3. Каждое свойство становится возможным столбцом (атрибутом) с тем же именем. Столбцы, соответствующие необязательным свойствам, могут содержать неопределенные значения; столбцы, соответст-

вующие обязательным свойствам, — не могут. Если свойство является множественным, то для него строится отдельное отношение.

4. Компоненты уникального идентификатора сущности превращаются в первичный ключ таблицы. Если имеется несколько возможных уникальных идентификаторов (потенциальных ключей), выбирается наиболее используемый. Если сущность слабая (зависимая), то в состав уникального идентификатора добавляются связи с соответствующими сильными (самостоятельными) сущностями — к множеству атрибутов первичного ключа добавляются копии уникальных идентификаторов сильных сущностей (этот процесс может продолжаться рекурсивно). Для именования этих атрибутов используются имена концов связей и/или имена сущностей.

5. Связи «многие к одному» и «один к одному» становятся внешними ключами. То есть создается копия уникального идентификатора с конца связи «один», и соответствующие атрибуты составляют внешний ключ.

6. Индексы создаются для первичного ключа (уникальный индекс), а также внешних ключей и тех атрибутов, которые будут часто использоваться в запросах.

7. Если в концептуальной модели присутствуют подтипы, то возможны два варианта. Все подтипы хранятся в одной таблице, которая создается для самого внешнего супертипа, а для подтипов создаются представления. В таблицу добавляется по крайней мере один атрибут, содержащий код типа, и он становится частью первичного ключа. Во втором случае для каждого подтипа создается отдельная таблица и для каждого подтипа первого уровня (для более нижних — представления) супертип воссоздается с помощью представления UNION (из всех таблиц подтипов выбираются общие атрибуты — атрибуты супертипа).

Рассмотрим некоторые варианты построения логических моделей по ER-диаграммам в зависимости от характера связей между сущностями.

Связи «один к одному»

Такая связь на логическом уровне реализуется как связь «один ко многим» или «многие ко многим». Отдельно рассматривается случай связи сильной и слабой (зависимой) сущности.

Если связь между сильной и слабой сущностью является обязательной с двух сторон (рис. 6.10), то на логическом уровне сущности

объединяются. В качестве первичного ключа при этом можно рекомендовать первичный ключ сильной сущности.

На рис. 6.10 изображен фрагмент ER-диаграммы, описывающей обязательное наличие у каждого сотрудника документа, удостоверяющего его личность. Каждому экземпляру сущности СОТРУДНИК должен соответствовать один экземпляр сущности ДОКУМЕНТ, которая является слабой (зависимой) и самостоятельно существовать в ПрО не может. Такой ER-диаграмме на логическом уровне соответствует отношение «Сотрудник» с первичным ключом *Табельный номер*:

Сотрудник (*Табельный номер*, Номер, ...)

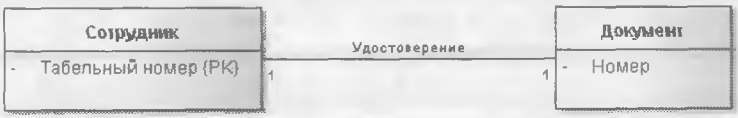


Рис. 6.10. Фрагмент ER-диаграммы для идентифицирующей связи «один к одному»

Если обе сущности, участвующие в связи, являются сильными, то одна из сущностей рассматривается как родительская, а другая — как дочерняя. К атрибутам дочерней сущности добавляется (в качестве внешнего ключа) первичный ключ родительской сущности. Выбор родительской сущности определяется характером участия в связи: для родительской сущности участие должно быть *обязательным*.

Например, фрагменту ER-диаграммы, представленному на рис. 6.11, на логическом уровне будет соответствовать совокупность двух отношений:

Сотрудник (*Табельный номер*, ...)
Автомобиль (*Номер техпаспорта*, Табельный номер, ...)

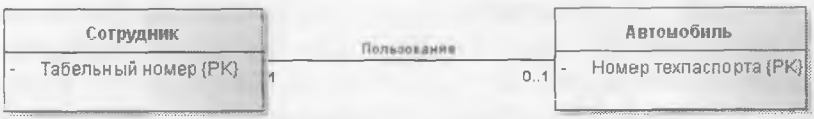


Рис. 6.11. Фрагмент ER-диаграммы с обязательным участием одной из сущностей

Атрибут *Табельный номер* добавлен в отношение АВТОМОБИЛЬ в роли внешнего ключа.

Связь, в которой участие обеих сущностей носит необязательный характер, может быть рассмотрена как связь «многие ко многим». На логическом уровне при этом формируется новая таблица (для сущно-

сти-связи), первичными ключами которой становятся первичные ключи обеих сущностей.

Фрагменту ER-диаграммы, представленному на рис. 6.12, на логическом уровне будет соответствовать совокупность трех отношений:

Сотрудник (*Табельный номер, ...*)
Автомобиль (*Номер техпаспорта, ...*)
Пользование (*Номер техпаспорта, Табельный номер, ...*)

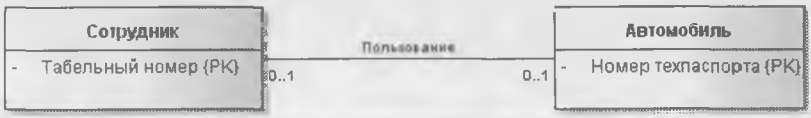


Рис. 6.12. Фрагмент ER-диаграммы с необязательным участием обеих сущностей

Связь «многие к одному» или «один ко многим»

В общем случае при реализации связи на логическом уровне уникальный идентификатор с конца связи «один» (родительской сущности) добавляется к атрибутам отношения с конца связи «многие» (дочерней сущности) и становится внешним ключом. Однако отдельно рассматриваются случаи присутствия в связи слабой сущности и необязательного участия родительской сущности в связи.

В случае присутствия слабой сущности (идентифицирующей связи) уникальный идентификатор сильной сущности включается в состав уникального идентификатора слабой сущности (рис. 6.13).

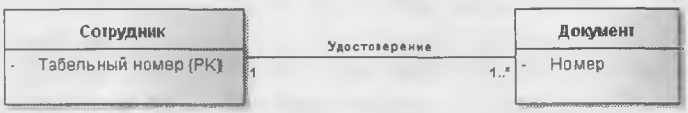


Рис. 6.13. Фрагмент ER-диаграммы для идентифицирующей связи «один ко многим»

Фрагменту ER-диаграммы, представленному на рис. 6.13, на логическом уровне будет соответствовать совокупность двух отношений:

Сотрудник (*Табельный номер, ...*)
Документ (*Табельный номер, Номер, ...*)

Атрибут *Номер* в составе отношения ДОКУМЕНТ выполняет две функции: входит в состав первичного ключа и является внешним ключом для связи с родительским отношением СОТРУДНИК.

Если обе сущности, участвующие в связи, являются сильными и со стороны родительской сущности носит обязательный характер, то к атрибутам дочерней сущности добавляется (в качестве внешнего ключа) первичный ключ родительской сущности.

Если участие родительской сущности в связи носит необязательный характер (рис. 6.14), то связь может рассматриваться как связь «многие ко многим» с образованием нового отношения.

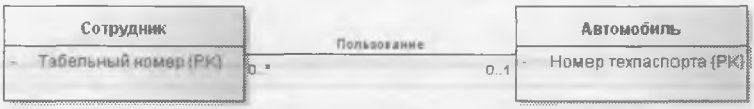


Рис. 6.14. Фрагмент ER-диаграммы для необязательной связи «один ко многим»

6.3. Постреляционная модель данных

Любое отношение по правилам реляционной модели должно находиться по крайней мере в первой нормальной форме (1НФ), т. е. все используемые домены отношения должны содержать только атомарные значения. Это снижает выразительность модели данных при описании целого ряда предметных областей, когда атрибут должен содержать множественные значения (например, перечень сотрудников, участвующих в проекте) или обладает внутренней структурой (например, атрибут *Контакты*, который может содержать несколько различных адресов и номеров телефонов).

В ряде случаев нормализованная реляционная модель предметной области несколько искусственна, поскольку в нее добавляются отношения, не присутствующие в самой предметной области как сущности (например, таблица, связывающая сотрудника с проектом). Кроме того, нормализация отношений требует разработки более сложных запросов с использованием нескольких таблиц.

Постреляционная (post-relational) модель данных была разработана с целью преодоления такого рода недостатков реляционной модели. Создание модели данных, допускающей неатомарность значений атрибутов кортежей, потребовало разработки новых правил нормализации. Основой нормализации в постреляционной модели данных служит так называемая «*не первая нормальная форма*» — НФ2[1] (Non First Normal Form — NF2), позволяющая атрибуту быть атомарным

[1] Не надо путать со второй нормальной формой, обозначаемой 2НФ.

(как в реляционной модели) или множественным. Множественный атрибут описывается вложенным отношением (множеством кортежей). Атрибуты такого вложенного отношения также в свою очередь могут быть множественными.

К числу достаточно известных постреляционных СУБД можно, например, отнести системы UniVerse (Vmark Software), Postgres (University of California at Berkeley), ADABAS (AG Software).

6.4. Объектно ориентированная модель данных

Появление объектно ориентированного (ОО) подхода в программировании внесло фундаментальные изменения во взгляды на данные и процедуры их обработки. Данные и процедуры традиционно хранились отдельно: данные и связи между ними — в БД, процедуры обработки — в прикладной программе. ОО подход позволяет хранить процедуры обработки вместе с данными, т. е. понятие сущности может быть расширено процедурами ее поведения в рамках предметной области.

В объектно ориентированном программировании базовыми единицами являются объекты. *Объект* — это понятие, сочетающее в себе совокупность данных и действий над ними. Каждый объект характеризуется своим состоянием, которое определяется текущими значениями его *атрибутов* (характеристических свойств). Атрибутами объекта могут быть не только атомарные величины, но и объекты.

Объект сохраняет свое состояние до тех пор, пока оно не будет изменено через вызов методов этого объекта. Это существенно ограничивает возможность несанкционированного или некорректного доступа к объекту. Объект может посылать *сообщения* другим объектам и принимать сообщения от них. Сообщение является совокупностью данных определенного типа, передаваемых объектом-отправителем объекту-получателю, имя которого указывается в сообщении. Получатель реагирует на сообщение выполнением некоторого метода.

Объекты с одинаковыми возможностями (данными и методами) объединяет термин *класс*. Класс описывает общее поведение и характеристики набора аналогичных друг другу объектов. Объект — это экземпляр класса или, другими словами, переменная, тип которой задается классом. Соотношения между объектом и классом аналогичны соотношениям между переменной и типом.

Базовыми механизмами объектно ориентированного подхода являются:

1) инкапсуляция (encapsulation) — объединение атрибутов и методов доступа к ним в одном объекте. Пользователю (в широком смысле) предоставляется только спецификация объекта (описание класса), а его реализация скрывается. В идеале доступ к атрибутам объекта возможен только через его методы;

2) наследование (inheritance). Наследования дочерним классом атрибутов и методов родительского класса. Допускается возможность добавления собственных атрибутов и методов, а также модификация некоторых унаследованных методов. Различают единичное (не более одного родительского класса) и множественное (несколько родительских классов) наследование;

3) полиморфизм (polymorphism). Способность различных объектов по-разному обрабатывать одинаковые сообщения. При этом различные объекты используют одинаковую абстракцию, т. е. могут обладать свойствами и методами с одинаковыми именами. Однако обращение к ним будет вызывать различную реакцию для различных объектов. На практике это означает, что можно создать общий интерфейс вызова для группы близких по смыслу действий;

4) абстракция (abstraction). Спецификация методов класса на уровне вызова (без реализации). Используется для объявления отдельных методов, которые должны быть реализованы в дочерних классах, а также для создания интерфейсов (interface). Интерфейс — это спецификация взаимодействия между объектами (поименованный перечень методов, которые должны быть реализованы в объекте).

Использование ОО-подхода при моделировании данных привело к созданию и развитию двух направлений — объектно-реляционного и «чистого» объектно ориентированного.

Объектно ориентированные БД являются навигационными: доступ к данным осуществляется с помощью связей, хранящихся в самих данных. Это означает, что запросы к такой БД должны ориентироваться на имеющиеся связи между объектами, т. е. заранее не запланированные связи не могут создаваться «на лету», как в реляционной модели.

Следует отметить, что многие производители современных СУБД стараются так или иначе реализовать в них принципы ОО-подхода. Так, например, компания Microsoft разработала для MS SQL Server компонент LINQ (Language-Integrated Query), позволяющий обращаться к записям таблиц базы данных как к объектам; компания

MySQL AB дополнила СУБД MySQL специальным средством Object-relational mapping tool.

Наиболее ярким представителем объектно-реляционных СУБД можно считать СУБД Oracle Database. Oracle Database позволяет объявлять классы, которые затем могут использоваться как домены при описании атрибутов таблицы либо служить описанием структуры таблицы в целом (атрибуты класса трактуются как атрибуты таблицы). В первом случае объектом является отдельное поле в записи, во втором случае — целая запись.

К «чистым» ОО СУБД можно отнести СУБД Caché. Объектная модель данных в ней основана на стандарте ODMG (Object Database Management Group — группа по объектному управлению базами данных). В СУБД реализованы все базовые механизмы ОО-подхода (инкапсуляция, наследование, полиморфизм, абстракция).

6.5. Технологии обработки данных на основе XML

Многие из проблем, с которыми связаны задачи интеграции данных, поддерживаемых средствами Internet, аналогичны проблемам создания систем неоднородных баз данных. К тому же множество Internet-источников велико и не постоянно, каждый из них в большой степени автономен и характеризуется своими метаданными.

Создание систем интеграции данных требует, как и в случае классических баз данных, выбора методики для моделирования предметной области. Однако, кроме модели самих информационных объектов, необходимо также иметь модель Internet (как среды доступа), определить структуру Web-сайтов и Web-страниц (как ресурса).

Важной особенностью моделирования Internet-ресурса является и то, что во многих случаях данные слабо структурированы: нет схемы, которая была бы задана заранее, а данные из разных источников могут различаться как набором атрибутов, так и иметь различные типы. В этом случае система должна формировать схему на основе получаемых метаданных, возможно, в момент получения самих данных, а язык манипулирования данными в результате обработки запроса должен позволять генерировать сложные структуры.

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 гибкой

структуры по сравнению с реляционными СУБД, но также требуют, чтобы все данные укладывались в заранее заданную схему.

При рассмотрении модели слабо структурированных данных обычно выделяют две основные проблемы: во-первых, структура данных обычно известна лишь частично. Набор данных может не иметь явно определенной схемы (schema-less), или схема может быть определена в теле документа — самоопределенный (self-describing) документ. Во-вторых, структура является вложенной или даже циклической, что требует от языков управления данными развитых рекурсивных возможностей. Реляционная алгебра этому требованию не соответствует, поэтому большинство подходов к управлению слабо структурированными данными основано на использовании языков запросов, обеспечивающих прохождение по древовидному размеченному графу, который служит для идентификации данных путем указания 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.

*Для интеграции слабо структурированных данных широко используется XML (*Extensible Markup Language*) — язык разметки, описывающий класс объектов данных, называемых XML-документами.*

XML изначально является средством информационного *обмена*[1]: XML-документ — это поток текстовых данных, которые может принимать и передавать приложение. XML практически не зависит от платформы, операционной системы, языка программирования и всех основных платформ, включая MS Windows и UNIX, поддержан стандартом, утвержденным World Wide Web Consortium (W3C).

XML представляет собой средство для описания грамматики представления и контроля правильности составления документов. Если XML-документ не нарушает правила синтаксиса XML, то он называется формально-правильным или хорошо оформленным *документом, и разборщики XML-документов будут работать с ним корректно.*

---

[1] В общем случае технология XML, помимо собственно языка разметки, для определения и обработки XML-документов использует ряд связанных стандартов, в том числе: Document Type Definitions (DTD) и XML-схемы, позволяющие определять XML-документы, спецификацию Namespaces, язык указания пути (XPath), язык указателей (XML Pointer Language — XPointer), язык ссылок (XML Linking Language — XLink), язык запросов (XQuery), объектную модель (Document Object Model — DOM), интерфейсы API, язык преобразований (Extensible Stylesheet Language — XSLT).

Состоятельными называются хорошо оформленные XML-документы, удовлетворяющие требованиям *DTD* (*Document Type Definition*) или *XML-схем*, определяющих его структуру и содержание. Главным достоинством схем является то, что они позволяют описывать правила для XML-документа средствами самого XML.

При обработке XML-документов XML-процессорами используется *объектная модель документа* (*DOM*) представляющая структуру и содержание документа в виде совокупности узлов, каждый из которых имеет свои свойства (имя, тип, значение, число дочерних узлов) и методы (создание, удаление, вставка).

Если разбираемый документ не соответствует объявленной схеме, процессор прекращает его разбор и сигнализирует об ошибке. Таким образом, схема накладывает ограничения на структуру данных.

XML-представление информации может быть реализовано как средствами «истинной» XML-СУБД (native XML database), которая разработана для работы с XML-данными, так и XML-расширениями существующих промышленных СУРБД, таких как IBM DB2, Oracle, Sybase, MySQL и Microsoft SQL Server. Хотя следует отметить, что XML-СУБД пока не нашли распространения, а при добавлении XML-функциональности в РСУБД становится непонятно, остаются ли они реляционными.

6.5.1. XML и реляционная модель данных

По аналогии с реляционной БД, если свойствам сущности соответствуют поля таблицы, то в XML-документе им могут соответствовать либо атрибуты, либо значения элементов, либо вложенные элементы. Рассмотрим типичную реляционную таблицу «Кадровый состав» (см. рис. 6.1).

Представим фрагмент этой таблицы в виде XML-документа с именем *Кадровый состав*, при этом атрибуты таблицы можно описать тремя способами:

1) в атрибутной форме, где им будут соответствовать атрибуты пустого элемента.

```
<Кадровый состав>
<Сотрудник Таб№='121' ФИО='Иванов И.И.'
     Должность ='Зав. каф.' Каф = '22'/>
<Сотрудник Таб№='231' ФИО='Сидоров С.С.'
     Должность ='Проф.' Каф = '22'/>
```

```
<Сотрудник Таб№='123' ФИО='Гиацинтова Г.Г..'
     Должность ='Проф.' Каф = '23'/>
</Кадровый состав >
```

2) в элементной форме, где все атрибуты будут вложенными элементами элемента, представляющего таблицу, к которой они относятся.

```
<Кадровый состав>
<Сотрудник>
<Таб№>121</Таб№>
<ФИО>Иванов И.И.</ФИО>
<Должность>Зав. каф.</Должность>
<Каф>22</Каф>
</Сотрудник >
<Таб№>231</Таб№>
<ФИО>Сидоров С.С.</ФИО>
<Должность>Проф.</Должность>
<Каф>22</Каф>
</Сотрудник >
<Таб№>123</Таб№>
<ФИО>Гиацинтова Г.Г.</ФИО>
<Должность>Проф.</Должность>
<Каф>23</Каф>
</Сотрудник>
</Кадровый состав>
```

3) в смешанной форме:

```
<Кадровый состав>
<Сотрудник Таб№='121' > Иванов И.И.
<Должность>Зав. каф.</Должность>
<Кафедра>22</Кафедра>
</Сотрудник >
<Сотрудник Таб№='231'> Сидоров С.С.
<Должность> Проф.</Должность>
<Кафедра> 22 </Кафедра>
</Сотрудник >
< Сотрудник Таб№='123'> Гиацинтова Г.Г.
<Должность> Проф.</Должность>
<Кафедра> 23 </Кафедра>
</Сотрудник>
</Кадровый состав>
```

Атрибутная форма представления требует меньших XML-потоков и работает эффективно при больших объемах данных, но у каждого элемента (и, следовательно, у каждого объекта) может быть только по одному значению каждого атрибута. Элементная форма полезна для потенциально многозначных характеристик.

6.5.2. Представление связей с помощью XML

Связи между таблицами в реляционных БД реализуются через значения атрибутов (например, посредством аппарата внешних ключей).

С помощью XML связь в простейшем случае может быть реализована в виде вложенных повторяющихся элементов. Например, для фрагментов таблиц «Структура» и «Кадровый состав» (рис. 6.1):

```
<Кадровый состав>
<Кафедра Каф='22' Телефон='25-15' Корпус='А' Ком='322' >
<Сотрудник Таб№='121' ФИО='Иванов И.И.'
     Должность ='Зав. каф.' />
<Сотрудник Таб№='231' ФИО='Сидоров С.С.'
     Должность ='Проф.' />
</Кафедра>
<Кафедра Каф='23' Телефон='38-42' Корпус='В' Ком='221' >
<Сотрудник Таб№='123' ФИО='Гиацинтова Г.Г..'
     Должность ='Проф.' />
</Кафедра>
</Кадровый состав>
```

6.6. Многомерная модель данных

Модель данных, поддерживающая OLAP-технологии, должна обеспечивать долговременное хранение сведений о некоторых свершившихся *фактах* для их последующей аналитической обработки. Структуру факта (в типовом отношении) образует набор атомарных *параметров*, каждый из которых в рамках конкретного факта принимает свое значение. Например, если в качестве факта рассматривать ежемесячный объем денежных средств, освоенных отдельным сотрудником в рамках работы над отдельным договором, то его структуру

составят следующие параметры: *№ Договора*, *Таб. № Сотрудника*, *Год*, *Месяц*, *Сумма*.

Все параметры факта делятся на *независимые* и *зависимые*. Независимые параметры в совокупности обеспечивают уникальность каждого факта, зависимые параметры определяются значением независимых. В приведенном примере независимые параметры — *№ Договора*, *Таб. № Сотрудника*, *Год*, *Месяц* и единственный зависимый параметр — *Сумма*.

Значения независимых параметров представлены шкалами *измерений* или *размерностей*.

С реляционной точки зрения такая модель данных всегда представима одной таблицей фактов и несколькими таблицами измерений — схема «звезда» (рис. 6.15).

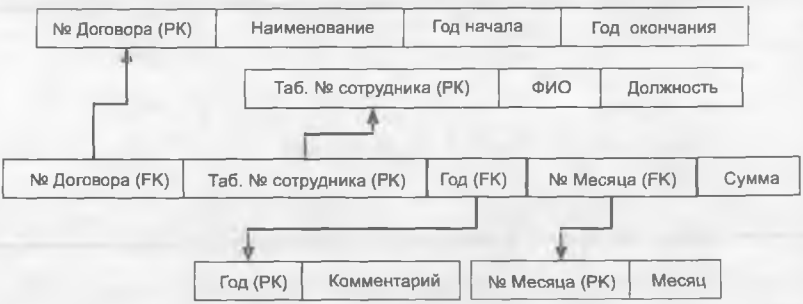


Рис. 6.15. Схема «звезда»

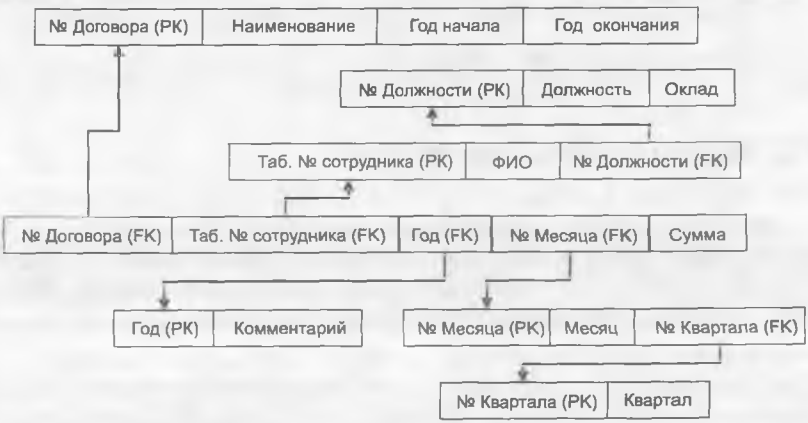


Рис. 6.16. Схема «снежинка»

Измерения могут быть выстроены в иерархии, например Квартал — Месяц, Должность — Сотрудник. В этом случае создается отдельная таблица для каждого уровня измерения — схема «снежинка» (рис. 6.16). Такая структура более сложная, но практически не обладает излишней избыточностью.

Выбор схемы в конкретном случае зависит от задач анализа данных.

6.7. Колоночные БД

Широкое использование реляционных СУБД, начало которого приходится на 60-е годы прошлого столетия, позволило внедрить автоматизированные системы, в основном учетного назначения, практически во все сферы человеческой деятельности. Архитектурные особенности реляционных СУБД, такие как хранение данных в виде последовательностей записей фиксированной длины, индексирование записей, журнализация операций, позволили сделать такие системы достаточно надежными и оперативными.

Системы оперативной поддержки бизнес-функций предметной области ориентированы на ввод, обработку и поиск отдельных записей или совокупности записей, удовлетворяющих заданному условию отбора. БД таких систем, ориентированные на логическую организацию и физическое хранение данных как последовательности записей, логически размещенных друг за другом, обеспечивают компактное хранение записи как совокупности составляющих ее атрибутов (полей).

Однако объемы накопленных данных поставили задачу использовать эти данные не только как оперативные, но и как ретроспективные для целей анализа и принятия решений (например, в сфере управления). Стали создаваться ретроспективные БД, основной задачей которых стало агрегирование и количественный анализ данных.

В п. 1.4.2 показано, что характер обработки данных в оперативных и аналитических БД существенно различается. Если транзакционные приложения выполняют частые и короткие операции в рамках обработки одной или нескольких записей, то аналитические приложения для того, чтобы получить ответ на запрос, должны обеспечить обработку большого объема данных в реальном масштабе времени. Различие операций состоит еще и в том, что транзакционные приложения по большей части работают с записью в целом, а аналитиче-

ские — с отдельным полем (или полями), для которого выполняются операции агрегирования, расчета суммарных или средних значений и т. п.

Одно из решений этой проблемы предложено в колоночных БД. Основная идея таких БД — преобразовывать каждое отношение логической модели в совокупность отношений, состоящих из одного атрибута исходного отношения и уникального идентификатора. Таким образом, с точки зрения реляционной модели в таких БД будет обеспечено физическое хранение таблиц «по колонкам» (отсюда и название — «колоночные БД»), а с точки зрения операций манипулирования данными может использоваться реляционное исчисление доменов.

Обработка аналитических запросов при таком способе организации данных оптимизируется за счет последовательной выборки данных для операций агрегирования. Однако операции, связанные с добавлением/удалением данных, существенно усложняются.

6.8. Темпоральные базы данных

Темпоральными данными называют такие, которые явно или неявно связаны с определенными моментами или промежутками времени.

Традиционные оперативные БД часто спроектированы таким образом, что любое изменение объекта в базе данных делает предыдущее состояние объекта недоступным. Основное назначение темпоральных БД (ТБД) — сохранить данные об *эволюции* объекта предметной области (в том числе от создания до завершения его существования). При каждом изменении состояния объекта в ТБД сохраняется его предыдущее состояние, связанное с записью о временном интервале, на котором это состояние объекта было актуальным; таким образом, в темпоральной базе данных для каждого факта существует промежуток времени, когда этот факт являлся истинным в предметной области. Например, одной из областей применения ТБД являются системы управления технологическими процессами.

Однако помимо представления времени с точки зрения ПрО существует представление времени с точки зрения БД (транзакционное время). Каждой записи базы данных можно сопоставить промежуток времени, когда она была актуальной в базе данных, т. е. промежуток времени между моментами добавления записи и ее удаления из базы

данных. При этом операция обновления, которая вносит изменения в запись, работает как составная операция удаления старой записи и добавления новой.

Рассмотрим следующий пример. Пусть в таблице ТБД хранятся данные о планируемых и выполняемых проектах — код проекта, руководитель, дата начала и дата завершения — и поддерживается как время ПрО, так и транзакционное время (рис. 6.17). Наличие поддержки времени ПрО обеспечит сохранение данных о смене руководителя проекта в соответствии с распоряжениями, а транзакционного — сохранит моменты внесения изменений в БД. В таблице на рис. 6.17 транзакционное время либо отстает от времени ПрО либо опережает его. Транзакционное время в этом случае позволит исправить ситуацию, если некорректные данные были уже использованы в каких-либо отчетах. Например, по результатам отчета, составленного 10 марта 2011 г., руководителем проекта П12.1 считался бы Иванов И.И., хотя в соответствии с данными ПрО проектом руководил уже Мальцев К.Н. (информация о смене руководителя проекта была внесена в БД позже 9 марта 2011 г.).

Проект	Руководитель проекта	Дата начала проекта	Дата завершения проекта	Время ПрО	Транзакционное время
П12.1	Самойлов Ю.В.	2011-02-01	2011-12-30	С 2010-12-15 по 2011-01-14	С 2010-12-20 по 2011-01-10
П12.1	Иванов И.И.	2011-02-01	2011-12-30	С 2011-01-15 по 2011-03-09	С 2010-01-11 по 2011-03-10
П12.1	Мальцев К.Н.	2011-02-01	2011-12-30	С 2011-03-10 по наст. время	С 2011-03-11 по наст. время

Рис. 6.17. Таблица с поддержкой действительного и транзакционного времени

Таким образом, интервалы транзакционного времени свидетельствуют о времени изменения данных или исправления ошибок, а интервалы времени ПрО — об изменении некоторых параметров моделируемой действительности.

Важной особенностью ТБД является то, что они должны обеспечивать ответы на запросы по времени. Например, произвести выборку всех состояний объекта на временном интервале $[t_1, t_2]$ или получить состояния всех объектов, которые были актуальны в момент времени t (срез по времени t). Следовательно, методы хранения темпоральных баз данных должны эффективно поддерживать такие запросы.

6.9. Преимущества и недостатки моделей

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 логической 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. 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).

Сетевые и иерархические структуры в основном ориентированы на то, чтобы связи между данными хранились вместе с самими данными. Такое объединение реализовалось, например, агрегированием данных (построением сложных понятийных структур и данных) или введением ссылочного аппарата, фиксирующего семантические связи, непосредственно в записи данных.

Табличная форма представления информации является наиболее распространенной и понятной. Кроме того, такие семантически более сложные формы, как деревья и сети, путем введения некоторой избыточности могут быть сведены к табличным. При этом связи между данными также будут представлены в форме двумерных таблиц.

Реляционный подход, в основе которого лежит принцип разделения данных и связей, обеспечивает, с одной стороны, независимость данных, а с другой — более простые способы реализации хранения и обновления.

Многомерные модели, коммерческие реализации которых появились в начале 1990-х гг. для поддержки технологий OLAP, представляют собой некоторое расширение модели универсальных отноше-

ний новыми операционными возможностями, обеспечивающими, в частности, необходимые для OLAP функции агрегирования данных. Таким образом, многомерные модели представляют собой особую разновидность реляционной модели.

К числу основных достоинств объектно ориентированной модели данных можно отнести близость к восприятию мира, свойственному человеку (как следствие — более естественные процессы анализа и проектирования) и отсутствие необходимости несколько искусственного деления системы на базу данных и программное обеспечение.

Среди основных недостатков объектно ориентированной модели данных можно выделить отсутствие математической основы и сложность перехода (смены парадигмы) к ОО-модели от простой и распространенной реляционной модели данных.

6.10. Документальные системы и интеграция моделей

Приведенные выше положения разрабатывались и действительно широко используются для баз данных хорошо структурированной информации. Однако уже сегодня одной из важнейших проблем становится обеспечение интеграции неоднородных информационных ресурсов, и в частности слабо структурированных данных. Необходимость ее решения связывается со стремлением к полноценной интеграции систем баз данных в среду Web-технологий. При этом уже недостаточно простого обеспечения доступа к базе данных традиционным способом «из-под» HTML-форм. Нужна интеграция на модельном уровне. И в этом случае проблема семантической интероперабельности информационных ресурсов сводится к задаче разработки средств и технологий, предусматривающих явную спецификацию метаданных для ресурсов слабо структурированных данных на основе традиционных технологий моделирования из области баз данных.

Именно на достижение этой цели направлены интенсивные разработки WWW-консорциумом языка XML и его инфраструктуры (фактически новой модели данных для этой среды), объектной модели документов и других средств, которые, как можно ожидать, в близкое время станут основой технологий управления информационными ресурсами. Это направление связано с другой глобальной проблемой — организацией распределенных неоднородных информацион-

ных систем на основе построения репозиториев метаданных (этому понятию в классических работах по проектированию баз данных соответствует понятие «словарь данных»), обеспечивающих возможность семантического отождествления ресурсов и, таким образом, возможность их целенаправленного повторного использования.

Контрольные вопросы

1. Перечислите логические модели на основе записей.
2. Перечислите и охарактеризуйте правила Кодда.
3. Дайте определение нормальной формы, которой, как минимум, должно удовлетворять каждое отношение.
4. Определите назначение методов нормализации данных.
5. Назовите типы аномалий, которые могут возникать в отношениях с избыточными данными. Приведите примеры.
6. Дайте определение понятия функциональной зависимости. Приведите примеры.
7. Сформулируйте понятие полной функциональной зависимости и покажите, как оно связано с 2НФ. Приведите пример отношения, не находящегося в 2НФ.
8. Сформулируйте понятие транзитивной зависимости и покажите, как оно связано с 3НФ. Приведите пример отношения, не находящегося в 3НФ.
9. Перечислите основные правила преобразования ER-диаграммы в реляционную модель данных.
10. Назовите базовые механизмы объектно ориентированного подхода.
11. Сформулируйте назначение XML.
12. Перечислите различия между схемами «звезда» и «снежинка» многомерной модели данных.

Глава 7
ПРИМЕР ПРОЕКТИРОВАНИЯ РЕЛЯЦИОННОЙ БАЗЫ ДАННЫХ

Задача проектирования БД для предметной области (по технологии нисходящего проектирования — см п. 4.6) состоит в том, чтобы обеспечить поддержку не только любых ныне используемых, но и будущих приложений. Таким образом, БД создают основу для обработки изменяющихся, заранее неизвестных запросов и создания приложений, для которых невозможно заранее определить требования к данным. Это позволяет в дальнейшем строить на основе предметных БД достаточно стабильные информационные системы, т. е. системы, в которых большинство изменений можно осуществить без переписывания старых приложений.

С другой стороны, основывая проектирование БД на реализации текущих и видимых задач (технология восходящего проектирования), можно существенно ускорить создание информационной системы, структура которой учитывает наиболее часто встречающиеся пути доступа к данным. Однако по мере увеличения количества таких информационных систем быстро увеличивается число прикладных БД и, соответственно, резко возрастает уровень дублирования данных, повышается стоимость их ведения.

Желание достичь одновременно гибкости и эффективности приводит к тому, что в общем случае предметный подход используется для построения первоначальной информационной структуры, а прикладной — для ее совершенствования с целью повышения эффективности обработки данных.

При проектировании информационной системы необходимо провести анализ целей этой системы и выявить требования к ней отдельных пользователей (см. главу 6). Сбор данных начинается с выявления и изучения объектов информационной среды и процессов, в

которых эти объекты участвуют. Объекты (сущности) группируются по типу и по мощности связей между ними (студент — сессия, преподаватель — дисциплина и т. д.).

Дальнейшая задача проектирования БД — это сокращение избыточности хранимых данных, а следовательно, экономия объема используемой памяти, уменьшение затрат на многократные операции обновления избыточных копий и устранение возможности возникновения противоречий из-за хранения в разных местах сведений об одном и том же объекте. Такой проект БД можно создать, используя методологию нормализации отношений.

Рассмотрим следующую задачу: пусть необходимо обеспечить сбор и обработку данных по результатам сдачи экзаменов и зачетов студентами факультета. Организация данных должна обеспечивать:

- выполнение текущего учебного плана;
- формирование ведомостей по отдельным дисциплинам для групп студентов;
- формирование листов зачетных книжек студентов;
- формирование сводной ведомости курса;
- расчет среднего балла по дисциплинам и т. п.

7.1. Восходящее проектирование (универсальное отношение)

Проектирование БД на основе описания предметной области в виде сводной таблицы (технология восходящего проектирования) предполагает выявление необходимого набора атрибутов — характеристических свойств объектов таким образом, чтобы каждый из этих атрибутов имел уникальное имя, а также представление этих атрибутов в виде двумерной таблицы. В общем случае такие сводные таблицы характеризуются тем, что на пересечении строки и столбца могут иметь более одного значения атрибута.

Для решения поставленной задачи представим описание предметной области в виде сводной таблицы, пример которой приведен на рис. 7.1. Предложенная таблица отражает результаты сдачи сессии (шкала оценок: 0 — незачет; 1 — зачет; 2, 3, 4, 5 — экзаменационная оценка).

Для преобразования сводной таблицы в отношение необходимо реструктурировать таблицу, например с помощью простого процесса вставки, результат которой показан на рис. 7.2.

№ зач. кн.	ФИО студента	Семестр	Дисциплина	Форма отчетности	Оценка	Кол-во часов	ФИО преподавателя	Должность	Кафедра
	Иванов В.П.	1	Английский язык	зачет	1	60	Цветкова А. Ю.	доцент	35
			Математический анализ	зачет	1	28	Рыбин К. К.	ст. преп.	22
			Математический анализ	экзамен	5	32	Раков И. И.	проф.	22
			Программирование	экзамен	5	32	Зайчиков А. А.	проф.	68
			Линейная алгебра	зачет	1	24	Волков Г. И.	доцент.	22
			Линейная алгебра	экзамен	4	28	Волков Г. И.	доцент	22
			История Отечества	экзамен	5	24	Москвин А. П.	проф.	30
	Петрова А.П.	1	Английский язык	зачет	1	60	Цветкова А. Ю.	доцент	35
			Математический анализ	зачет	1	28	Рыбин К. К.	ст. преп.	22
			Математический анализ	экзамен	3	32	Раков И. И.	проф.	22
			Программирование	экзамен	4	32	Зайчиков А. А.	проф.	68
			Линейная алгебра	зачет	1	24	Волков Г. И.	доцент	22
			Линейная алгебра	экзамен	4	28	Волков Г. И.	доцент	22
			История Отечества	экзамен	5	24	Москвин А. П.	проф.	30

Рис. 7.1. Исходные данные для создания БД «Сессия»

№ зач. кн.	ФИО студента	Семестр	Дисциплина	Форма отчетности	Оценка	Кол-во часов	ФИО преподавателя	Должность	Кафедра
	Иванов В. П.	1	Английский язык	зачет	1	60	Цветкова А. Ю.	доцент	35
	Иванов В. П.	1	Математический анализ	зачет	1	28	Рыбин К. К.	ст. преп.	22
	Иванов В. П.	1	Математический анализ	экзамен	5	32	Раков И. И.	проф.	22
	Иванов В. П.	1	Программирование	экзамен	5	32	Зайчиков А. А.	проф.	68
	Иванов В. П.	1	Линейная алгебра	зачет	1	24	Волков Г. И.	доцент	22
	Иванов В. П.	1	Линейная алгебра	экзамен	4	28	Волков Г. И.	доцент	22
	Иванов В. П.	1	История Отечества	экзамен	5	24	Москвин А. П.	проф.	30
	Петрова А.П.	1	Английский язык	зачет	1	60	Цветкова А. Ю.	доцент	35
	Петрова А.П.	1	Математический анализ	зачет	1	28	Рыбин К. К.	ст. преп.	22
	Петрова А.П.	1	Математический анализ	экзамен	3	32	Раков И. И.	проф.	22
	Петрова А.П.	1	Программирование	экзамен	4	32	Зайчиков А. А.	проф.	68
	Петрова А.П.	1	Линейная алгебра	зачет	1	24	Волков Г. И.	доцент	22
	Петрова А.П.	1	Линейная алгебра	экзамен	4	28	Волков Г. И.	доцент	22
	Петрова А.П.	1	История Отечества	экзамен	5	24	Москвин А. П.	проф.	30

Рис. 7.2 Универсальное отношение «Сводная ведомость»

Таблица на рис. 7.2 представляет собой корректное отношение. Такое отношение называют *универсальным* отношением проектируемой БД. В одно универсальное отношение включаются все представляющие интерес атрибуты, и оно может содержать все данные, которые предполагается размещать в БД в будущем. При проектировании некоторых БД универсальное отношение может использоваться в качестве отправной точки.

Однако очевидно, что подобные преобразования приводят к возникновению большого объема избыточных данных и вызывают проблемы (аномалии обновления), приводящие к нецелостности и противоречивости данных.

Для ликвидации избыточности и потенциальной противоречивости применяют аппарат нормализации (см. п. 6.2.3).

Применим правила нормализации к универсальному отношению «Сводная ведомость»:

```
Сводная ведомость (№ зачетной книжки, ФИО студента,
                   Дисциплина, Семестр, Форма отчетности,
                   Количество часов, Оценка, Дата сдачи,
                   ФИО преподавателя, Должность
                   преподавателя, Кафедра).
```

1. *Определение первичного ключа таблицы.*

Предположим, что каждый студент сдает один раз экзамен (зачет) по дисциплине учебного плана и получает оценку. Дисциплина учебного плана однозначно характеризуется наименованием, номером семестра, за который отчитывается студент, и формой отчетности (так как учебный план предусматривает сдачу и экзамена, и зачета по одной и той же дисциплине в рамках одного семестра). Тогда в качестве первичного ключа отношения «Сессия» можно использовать следующий набор атрибутов:

```
№ зачетной книжки, Дисциплина, Семестр, Форма
отчетности
```

2. *Выявление атрибутов, функционально зависящих от части составного ключа.*

Каждый из атрибутов — ФИО преподавателя, Должность преподавателя, Кафедра и Количество часов — функционально зависит только от атрибутов *Дисциплина, Семестр* и *Форма отчетности*, т. е. этот атрибут вместе с совокупностью атрибутов первичного клю-

ча (по первому правилу процедуры нормализации) составит вторую таблицу:

Учебный план (*Дисциплина, Семестр, Форма отчетности, Количество часов, ФИО преподавателя, Должность преподавателя, Кафедра*)

Из исходной таблицы при этом удаляются атрибуты ФИО преподавателя, Должность преподавателя, Кафедра и Количество часов:

Сводная ведомость (*№ зачетной книжки, Дисциплина, Семестр, Форма отчетности,* ФИО студента, Оценка).

Составной первичный ключ, повторяющийся в обеих таблицах, приводит к избыточности при дублировании информации сразу трех столбцов, поэтому кажется целесообразным ввести дополнительный атрибут — № (порядковый номер) — в таблицу «Учебный план» и использовать именно его в качестве первичного ключа. Тогда таблицы примут следующий вид:

Учебный план (*№ Дисциплины*, Дисциплина, Семестр, Форма отчетности, Кол-во часов, ФИО преподавателя, Должность преподавателя, Кафедра)

Сводная ведомость (*№ зачетной книжки, № Дисциплины,* ФИО студента, Оценка).

В таблице «Сводная ведомость» осталась еще одна частичная функциональная зависимость: *№ зачетной книжки → ФИО студента.* Ликвидировав эту зависимость, получим еще одну таблицу — «Студент»:

Студент (*№ зачетной книжки,* ФИО студента)

Ликвидация функциональной зависимости приведет к изменению таблицы «Сводная ведомость»:

Сводная ведомость (*№ зачетной книжки, № Дисциплины,* Оценка)

3. *Выявление транзитивных зависимостей*

В таблице «Учебный план» выявляются следующие транзитивные зависимости:

№ Дисциплины → Должность преподавателя

(*№ Дисциплины* → *ФИО преподавателя*; *ФИО преподавателя* → *Должность преподавателя*) и

№ Дисциплины → *Кафедра*;

(*№ Дисциплины* → *ФИО преподавателя*; *ФИО преподавателя* → *Кафедра*).

Применив второй шаг процедуры нормализации, получим таблицу «Кадровый состав»:

```
Кадровый состав (ФИО преподавателя, Должность
                 преподавателя, Кафедра)
```

Следует иметь в виду, что в этом случае в рамках предметной области считается, что атрибут *ФИО преподавателя* однозначно (уникально) характеризует конкретного преподавателя.

Итак, получили следующую декомпозицию универсального отношения:

```
Учебный план (№ Дисциплины, Дисциплина, Семестр,
              Форма отчетности, Кол-во часов,
              ФИО преподавателя)
Студент (№ зачетной книжки, ФИО студента)
Кадровый состав (ФИО преподавателя, Должность
                 преподавателя, Кафедра)
Сводная ведомость (№ зачетной книжки, № Дисциплины,
                   Оценка)
```

Такой декомпозиции достаточно для того, чтобы преобразовать исходное универсальное отношение к совокупности нормализованных таблиц (все полученные таблицы приведены к 3НФ и к НФБК).

7.2. Нисходящее проектирование

Рассмотрим процесс проектирования базы данных для решения поставленных задач с использованием технологии нисходящего проектирования.

7.2.1. Построение инфологической модели

Представим предметную область как взаимодействие двух сущностей — «Дисциплина учебного плана» и «Студент»: каждый студент сдает экзамен или зачет по некоторой дисциплине учебного

плана и получает оценку, которая должна быть зафиксирована в модели данных.

«Дисциплина учебного плана» с точки зрения решаемой задачи должна быть представлена группой свойств, позволяющих характеризовать ее в рамках каждого отдельного семестра: наименование дисциплины, семестр, количество часов, форма отчетности (экзамен или зачет) и данные о преподавателе, читающем дисциплину. Необходимость задания таких свойств обусловлена, с одной стороны, задачей организации хранения результатов сдачи экзаменов и зачетов (наименование дисциплины, семестр и форма отчетности), и с другой стороны — задачей формирования листов зачетных книжек (количество часов и данные о преподавателе). Отдельный экземпляр такой сущности однозначно идентифицируется тройкой свойств: наименование дисциплины, семестр и форма отчетности.

Сущность «Студент» для обеспечения выполнения объявленных функций должна характеризоваться следующими свойствами: номер зачетной книжки, фамилия, имя, отчество и номер группы. Уникальным идентификатором отдельного экземпляра сущности в данной предметной области может служить номер зачетной книжки, который остается неизменным на протяжении всего обучения.

Определим для сущности «Студент» еще два дополнительных свойства, которые не будут непосредственно обеспечивать решение поставленной задачи, но могут служить для реализации дополнительных (сервисных) функций (например, организации почтовой или телефонной связи): домашний адрес и номер телефона. Свойство «Домашний адрес», являясь по сути составным, будет на самом деле рассматриваться в контексте решаемых задач как простое, а свойство «Номер телефона» — как условное.

Взаимодействие сущностей реализуется связью «Сводная ведомость», т. е. Студент сдает экзамен (зачет) по Дисциплине учебного плана. Мощность связи — «многие ко многим» (M:M). Можно считать связь необязательной с обеих сторон, так как фиксируются только факты сдачи экзамена или зачета, а студент (до факта отчисления) может не явиться ни на один экзамен. Для идентификации связи отдельных экземпляров сущностей необходимо наличие у связи следующих дополнительных свойств: оценка и дата сдачи экзамена (зачета).

ER-диаграмма рассматриваемой задачи представлена в нотации диаграммы классов UML (рис. 7.3).

Сущности построенной ER-диаграммы отвечают требованиям *первой нормальной формы*, так как свойства их атомарны и определены

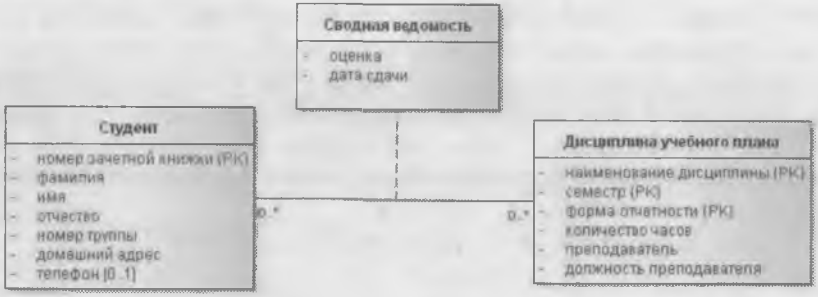


Рис. 7.3. ER-диаграмма рассматриваемой предметной области

первичные ключи. Однако при рассмотрении свойств сущности «Дисциплина учебного плана» можно заметить, что свойство «Должность преподавателя» транзитивно зависит от первичного ключа (через свойство «Преподаватель». Такая ситуация вызовет на этапе логического проектирования появление отношения, не находящегося в 3НФ. Чтобы ликвидировать транзитивную зависимость, выделим свойство «Преподаватель» в отдельную сущность «Кадровый состав».

Новая сущность характеризуется группой основных свойств: фамилия, имя, отчество, кафедра, должность — и группой дополнительных свойств: домашний адрес и телефон. Для сущности «Кадровый со-

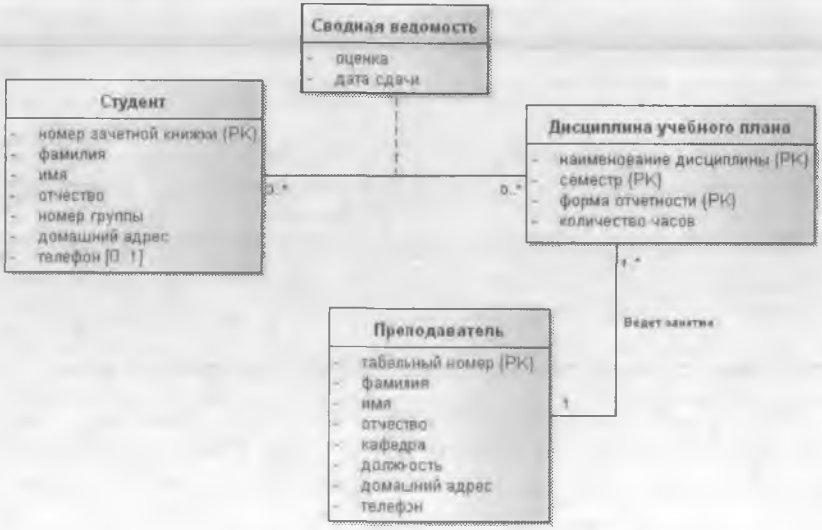


Рис. 7.4. Нормализованная ER-диаграмма

став» в качестве первичного ключа можно использовать свойство «Табельный номер», уникально характеризующее каждого сотрудника.

Взаимодействие новой сущности с сущностью «Дисциплина учебного плана» осуществляется посредством новой связи «Ведет занятия». Мощность связи — «многие к одному» (M : 1), т. е. несколько дисциплин учебного плана может читать один преподаватель. Связь является обязательной с обеих сторон.

Новый вариант ER-диаграммы представлен на рис. 7.4.

7.2.2. Построение реляционной схемы

Следующий этап проектирования — построение логической (даталогической) модели. В рассматриваемом случае задача этого этапа — преобразование ER-диаграммы в реляционную схему (см. п. 6.2.6).

Первые шаги преобразования состоят в превращении каждой сущности в отношение (таблицу). Связь типа «многие ко многим» (M : M), которую называют «сущность—связь», тоже превращается в отдельное отношение. Каждое свойство становится атрибутом — столбцом соответствующей таблицы.

После реализации этих шагов получаем реляционную схему (рис. 7.5), где представлены таблицы «Студент», «Учебный план», «Кадровый состав» и «Сводная ведомость», отображающие соответственно сущности «Студент», «Дисциплина учебного плана» и «Кадровый состав» и сущность-связь «Сводная ведомость».

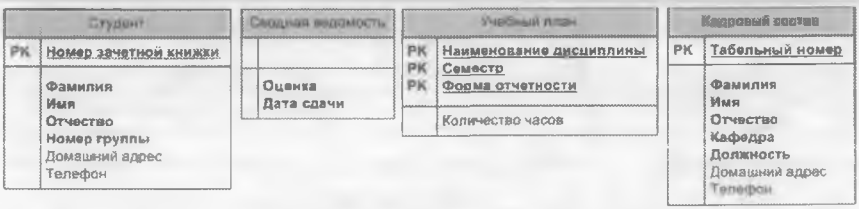


Рис. 7.5. Реляционная схема после первого этапа преобразований

Далее необходимо преобразовать связи во внешние ключи. Связь «многие ко многим», реализуемая отношением «Сводная ведомость», должна содержать уникальные идентификаторы сущностей — участников связи, которые в совокупности образуют первичный ключ отношения.

Рассмотрим первичный ключ отношения «Студент» — *Номер зачетной книжки*. Этот атрибут носит характер классификационного шифра и в общем случае не может быть представлен числовым значением, поэтому (в целях оптимизации связи) имеет смысл добавить в отношение «Студент» суррогатный первичный ключ — атрибут *ID Студент*.

Для однозначной идентификации дисциплины необходимо добавить в отношение «Сводная ведомость» атрибуты первичного ключа *Наименование*, *Семестр* и *Форма отчетности*. Для ликвидации избыточности и потенциальной противоречивости данных (например, если при переносе дисциплины на другой семестр обновить только строку таблицы «Учебный план», то содержимое таблицы «Сводная ведомость» станет не актуальным) добавим в отношение «Учебный план» суррогатный первичный ключ *ID План*, содержимое которого будет однозначно идентифицировать каждую строку таблицы.

Теперь для реализации связи «многие ко многим» в отношение «Сводная ведомость» добавим атрибуты *ID Студент* и *ID План*, каждый из которых входит в состав первичного ключа и представляет собой атрибут внешнего ключа для связи с соответствующей родительской таблицей.

Связь «Ведет занятия» предполагает добавление в таблицу «Учебный план» столбца *Табельный номер*. Реляционная схема со связями представлена на рис. 7.6.

Рассмотрим подробнее таблицу «Учебный план», которая содержит перечень дисциплин текущего учебного плана. Первичным ключом таблицы служит столбец *ID План*, который однозначно характеризует каждую дисциплину учебного плана с точностью до семестра, т. е. для дисциплин, протяженность изучения которых более одного семестра, в таблице будет отведено столько строк, сколько семестров длится изучение дисциплины. Тогда хранение наименований дисциплин в таблице «Учебный план» становится избыточным: например, если изучение английского языка длится 6 семестров, то наименование «Английский язык» будет повторено в 6 записях и есть вероятность сделать 6 различных ошибок при вводе одного и того же наименования.

Чтобы избежать этого, проведем декомпозицию отношения «Учебный план», выделив наименования дисциплин в отдельное отношение. В результате получим дополнительную таблицу «Дисциплина» со столбцами *ID Дисциплина* и *Наименование*, а столбец *Наименование* в таблице «Учебный план» заменим столбцом *ID Дисциплина*,

сформировав тем самым вторичный ключ, связывающий новую таблицу с таблицей «Учебный план».

Рис. 7.6. Реляционная схема со связями

Реляционная схема базы данных «Сессия» представлена следующими пятью таблицами:

«Студент» — содержит по одной строке для каждого из студентов;

«Учебный план» — содержит по одной строке для отдельной дисциплины отдельного семестра;

«Дисциплина» — содержит по одной строке для наименования дисциплины;

«Сводная ведомость» — содержит по одной строке для каждого результата сдачи отдельным студентом отдельной дисциплины;

«Кадровый состав» — содержит по одной строке для каждого из преподавателей.

На рис. 7.7 в графической форме изображены перечисленные таблицы, их столбцы, первичные и внешние ключи. Задание первичных и внешних ключей сопровождается построением дополнительных структур — индексов, обеспечивающих быстрый доступ к данным через значение ключа.

Рис. 7.7. Схема данных в СУБД MS Access

Таблицы «Сводная ведомость», «Дисциплины», «Кадровый состав» находятся в третьей нормальной форме (3НФ). Таблицы «Студент» и «Учебный план» находятся в нормальной форме Бойса — Кодда (НФБК), так как каждая из них имеет более одного потенциального ключа (например, в таблице «Учебный план» потенциальными ключами служат атрибут *ID План* и совокупность атрибутов *ID Дисциплина*, *Семестр* и *Форма отчетности*).

Следующий этап проектирования — определение доменов (типов) данных, хранящихся в столбцах таблиц. Параллельно с заданием типа необходимо сформулировать ограничения целостности, связанные с типом — перечень допустимых значений типа.

Исходя из особенностей данных и их функционального назначения требуется задать способ представления и границы возможных изменений для каждого из столбцов таблиц. При этом необходимо ответить на вопрос, данные каких типов должны храниться в столбцах и какова их максимальная длина (например, если в столбце предполагается хранить процентные значения, то достаточно будет целого типа данных длиной 1 байт, так как диапазон возможных значений от 0 до 100; если для данных столбца выбирается тип «строка символов», то желательно указать максимальный размер данных столбца и т. п.).

Далее, в каждой таблице должны быть выделены столбцы, которые *обязательно должны быть заполнены* при создании отдельной

строки таблицы. Задание такого ограничения целостности не позволит, например, ввести в таблицу «Студент» строку, в которой не указан номер группы. Если подобные ограничения целостности не будут заданы, в таблице могут появиться строки, которые не будут учтены при выполнении функций по обработке данных: появление в таблице «Студент» строки без номера группы приведет к ошибке при формировании ведомости.

Следующий важный момент — задание для столбцов значений по умолчанию. Значение по умолчанию впоследствии будет автоматически вводиться в указанный столбец для каждой строки таблицы. Например, в столбец *Дата сдачи* таблицы «Сводная ведомость» при заполнении очередной строки может автоматически заноситься текущая дата.

В табл. 7.1—7.5 приведены структуры таблиц базы данных «Сессия» с типами данных столбцов и предлагаемыми ограничениями целостности.

Таблица 7.1. **Структура таблицы «Студент»**

Наименование столбца	Тип данных	Ограничения
ID Студент	Целое число	Первичный ключ
Фамилия	Строка символов размером 30	Значение не должно быть пустым
Имя	Строка символов размером 15	Значение не должно быть пустым
Отчество	Строка символов размером 20	Значение не должно быть пустым
Номер группы	Строка символов размером 6	Значение не должно быть пустым
Адрес	Строка символов размером 30	
Телефон	Строка символов размером 15	

Таблица 7.2. **Структура таблицы «Дисциплина»**

Наименование столбца	Тип данных	Ограничения
ID Дисциплина	Целое число	Первичный ключ
Наименование	Строка символов размером 30	Значение уникально

Таблица 7.3. **Структура таблицы «Кадровый состав»**

Наименование столбца	Тип данных	Ограничения
Табельный номер	Целое число	Первичный ключ
Фамилия	Строка символов размером 30	Значение не должно быть пустым
Имя	Строка символов размером 15	Значение не должно быть пустым
Отчество	Строка символов размером 20	Значение не должно быть пустым
Должность	Строка символов размером 20	Значение не должно быть пустым
Кафедра	Строка символов размером 30	Значение не должно быть пустым
Адрес	Строка символов размером 30	
Телефон	Строка символов размером 15	

Таблица 7.4. **Структура таблицы «Учебный план»**

Наименование столбца	Тип данных	Ограничения
ID План	Целое число	Первичный ключ
ID Дисциплина	Целое число	Внешний ключ, значение не должно быть пустым
Семестр	Целое число	Значение не должно быть пустым и должно находиться в интервале от 1 до 10
Форма отчетности	Символ	
Количество часов	Целое число	
Табельный номер	Целое число	Внешний ключ, значение не должно быть пустым

Таблица 7.5. **Структура таблицы «Сводная ведомость»**

Наименование столбца	Тип данных	Ограничения
ID Студент	Целое число	Первичный ключ, внешний ключ
ID План	Целое число	Первичный ключ, внешний ключ
Оценка	Целое число	Значение не должно быть пустым и должно находиться в интервале от 0 до 5
Дата сдачи	Дата-время	Значение не должно быть пустым, по умолчанию — текущая дата

Контрольные задания и вопросы

1. Приведите примеры дополнительных функций, которые могут быть реализованы с помощью таблиц БД «Сессия».
2. Проведите декомпозицию отношения «Кадровый состав», выделив отношение «Штатное расписание».
3. Проведите декомпозицию отношения «Кадровый состав», выделив отношение «Структура факультета».
4. Какие изменения должны быть внесены в структуру БД «Сессия» для реализации функции назначения стипендии?

Глава 8
УПРАВЛЕНИЕ РЕЛЯЦИОННЫМИ БАЗАМИ ДАННЫХ

Управление базами данных осуществляется средствами лингвистического обеспечения СУБД. Внутренний язык СУБД для работы с данными обязательно включает два компонента:

1) *язык определения данных* (Data Definition Language — DDL), в состав которого входят средства описания базы данных на структурном уровне;

2) *язык манипулирования данными* (Data Manipulation Language — DML), реализующий основные возможности СУБД по поиску и обновлению хранимых данных.

Языки для работы с базами данных обычно не содержат средств для программирования условных операторов, операторов перехода или операторов цикла, существующих в языках программирования высокого уровня, поэтому их называют *подъязыками* баз данных.

Во многих СУБД предусмотрена возможность внедрения операторов подъязыка данных в программы, написанные на *базовых* языках программирования высокого уровня (например, Fortran, Pascal, Ada, C++, Java, Visual Basic). Некоторые подъязыки обеспечены интерпретаторами, которые поддерживают средства интерактивного выполнения операторов, вводимых непосредственно с терминала.

8.1. Язык определения данных

Схема базы данных описывается набором инструкций специального языка определения данных — DDL. DDL используется и как средство определения новых объектов БД, и как средство модификации существующих.

В состав языка определения реляционных данных должны входить возможности для описания отношений, их атрибутов, первичных и внешних ключей и других ограничений целостности.

Результатом компиляции DDL-операторов является набор описаний структур таблиц БД и ограничений целостности, хранимый в *системном каталоге*. Назначение системного каталога — интегрировать *метаданные*, т. е. данные, описывающие объекты базы данных. Метаданные позволяют формализовать способ доступа к самим данным и управление ими: перед доступом к реальным данным СУБД обычно обращается к системному каталогу.

Языки определения данных могут быть реализованы в вербальной форме и графическими средствами. Примером вербальной формы служат инструкции управления таблицами SQL. Графические средства обычно реализованы в модулях управления данными СУБД и представляют собой *конструкторы* для формирования структуры таблиц и задания ограничений целостности.

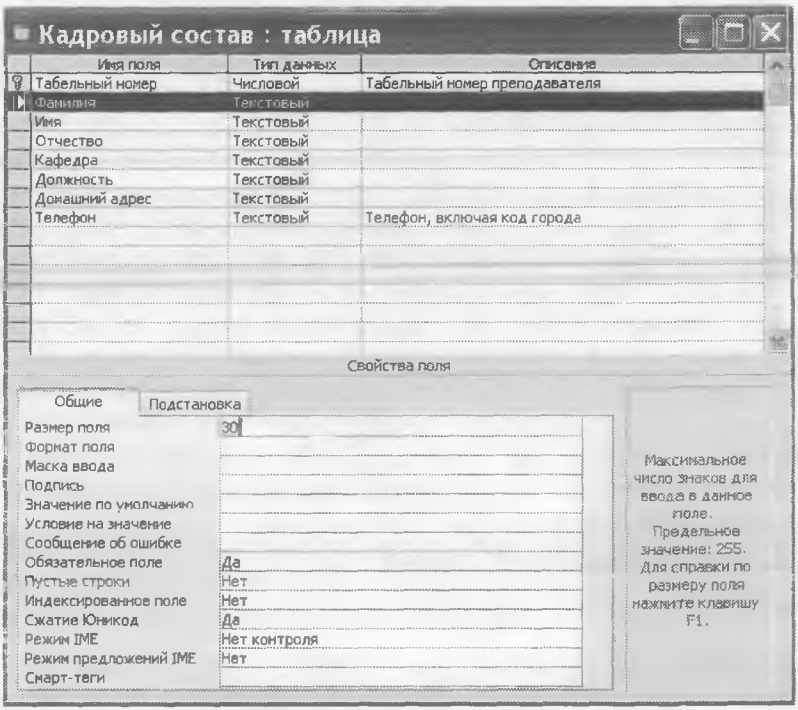


Рис. 8.1. Конструктор таблиц СУБД MS Access

На рис. 8.1 изображено окно Конструктора для создания структуры таблиц в СУБД MS Access. В нижней части окна вводятся ограничения целостности для выделенного в верхней части атрибута («Свойства поля»). Атрибуты, входящие в состав первичного ключа, идентифицируются пиктограммой «ключ». Окно Конструктора используется также и для внесения изменений в структуры таблиц — добавление или удаление атрибутов, изменение свойств атрибутов.

Окно создания и корректировки внешних ключей (ограничений целостности на уровне БД) показано на рис. 8.2 (графическая схема данных со связями представлена на рис. 7.7).

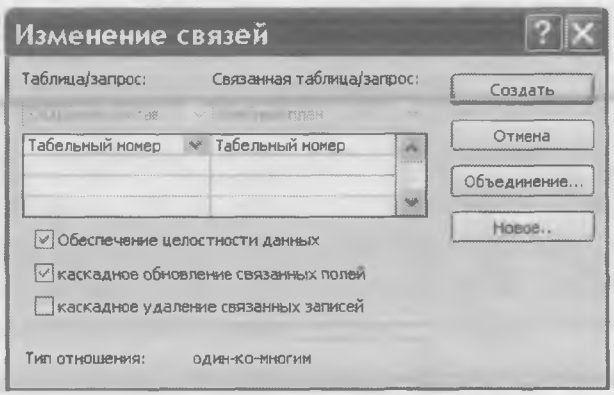


Рис. 8.2. Задание внешних ключей в СУБД MS Access

8.2. Язык манипулирования данными

Язык манипулирования данными представляет собой множество инструкций для поддержки основных операций управления данными БД.

К основным операциям управления данными относятся:

- вставка новых данных;
- модификация хранимых данных;
- поиск данных;
- удаление данных.

Поддержка языка манипулирования данными — одна из основных функций СУБД, обеспечивающая взаимодействие пользователя с БД на уровне управления данными.

Рассматривают два типа DML: *процедурный* и *непроцедурный*. Процедурные языки с помощью операторов описывают алгоритм формирования результата (т. е. то, *как* можно получить результат). Операторы непроцедурных языков описывают форму и содержание результата (т. е. то, *какой* результат должен быть получен). Таким образом, языки DML имеют разные базовые конструкции извлечения данных.

Процедурный язык манипулирования данными — это язык, который позволяет не только описать необходимые данные, но и точно указать, *как* их можно извлечь. Это значит, что для получения требуемых данных необходимо указать все операции доступа к данным (посредством вызова соответствующих процедур), которые должны быть выполнены. Обычно такой язык должен позволять последовательно извлечь отдельную запись; обработать ее; принять решение о том, поместить ли запись в формируемый результат; в зависимости от принятого решения добавить (или нет) запись в результат и далее продолжать процесс извлечения и обработки данных до тех пор, пока не будут обработаны все запрашиваемые данные. Для выполнения таких задач 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, так как должны поддерживать *навигационный* способ выбора данных.

Непроцедурный язык манипулирования данными — это язык, который дает возможность указать лишь *то, какие* данные должны быть получены (не описывая алгоритма, *как* их следует извлекать), поэтому непроцедурные языки часто также называют *декларативными языками*. Операторы непроцедурных DML позволяют определить весь набор необходимых данных и содержат компоненты, указывающие, какие данные должны войти в результат, какие объекты должны участвовать в формировании результата и по какому логическому условию необходимо проводить отбор. Способ получения данных непроцедурные DML не определяют: СУБД по собственным правилам транслирует выражение на языке манипулирования данными в процедуру (или набор процедур), которая обеспечивает необходимый результат. Такой подход в определенной степени обеспечивает независимость от данных — подробностей внутренней реализации структур данных и особенностей алгоритмов их извлечения и возможного преобразования.

Поддержку непроцедурных языков манипулирования данными в той или иной форме обычно обеспечивают реляционные СУБД — чаще всего это язык структурированных запросов SQL (Structured Query Language) или язык запросов по образцу QBE (Query by Example).

8.3. SQL

Изначально создаваемый как инструмент для выборки и представления данных, содержащихся в базе данных, SQL сегодня представляет собой нечто гораздо большее. Несмотря на то что выборка данных по-прежнему остается одной из наиболее важных функций SQL, сейчас этот язык используется для реализации всех функциональных возможностей, необходимых для управления БД, в том числе:

- *организации данных* — SQL позволяет определять и изменять структуру представления данных, а также устанавливать связи и ограничения целостности;
- *обработки данных* — SQL позволяет изменять содержимое базы данных: добавлять новые данные, удалять или обновлять уже имеющиеся в ней данные;
- *управления доступом* — SQL позволяет ограничивать возможности пользователя по чтению и изменению данных (защита данных от несанкционированного доступа) и координировать их совместное использование пользователями, работающими параллельно.

В стандартном SQL89, реализованном в полном объеме для реляционных СУБД, нет операторов проверки условий и ветвления, перехода, организации циклов и т. д. Однако SQL большинства промышленных СУБД содержит эти и многие другие операторы, позволяющие создавать полноценные процедуры обработки данных. Таким образом, хотя SQL и не объявляется как полноценный язык программирования, он является достаточно полным и мощным языком для управления взаимодействием с СУБД.

SQL является декларативным *подъязыком*, предназначенным для управления базами данных. Несмотря на не совсем точное название, SQL на сегодняшний день является *единственным* стандартным языком для работы с реляционными базами данных.

Операторы SQL встраиваются в базовый язык и дают возможность получать доступ к базам данных из прикладных программ. Кро-

ме того, из многих языков программирования операторы SQL можно посылать СУБД в явном виде, используя *интерфейс вызовов функций*.

Официальный стандарт языка SQL был опубликован в 1986 г. Американским институтом национальных стандартов (ANSI) и Международной организацией по стандартам (International Standards Organization — ISO), а в 1989 и 1992 гг. значительно расширен. Стандарт X/OPEN для переносимой среды программирования на основе операционной системы UNIX также включает в себя SQL в качестве языка для доступа к базам данных. Консорциум поставщиков компьютерного оборудования и баз данных (SQL Access Group) определил для SQL стандартный интерфейс вызовов функций, который является основой протокола ODBC компании Microsoft и входит также в стандарт X/OPEN. Эти стандарты de facto являются официальным одобрением SQL, и именно они ускорили завоевание им рынка.

Многие из членов комитетов по стандартизации ANSI и ISO представляли фирмы — поставщики различных СУБД, в каждой из которых был реализован собственный диалект SQL. Как и диалекты человеческого языка, диалекты SQL были в основном похожи друг на друга, однако несовместимы в деталях. Во многих случаях комитет просто игнорировал существующие различия и не стандартизировал некоторые части языка, определив, что они реализуются по усмотрению разработчика. Этот подход позволил объявить большое число реализаций SQL совместимыми со стандартом, однако сделал сам стандарт относительно слабым.

Чтобы заполнить эти пробелы, комитет ANSI продолжил свою работу и создал проект нового, более жесткого стандарта SQL2. В отличие от стандарта 1989 г., проект SQL2 предусматривал возможности, выходящие за рамки уже существующих в реальных коммерческих продуктах. Перечислим отличия SQL2.

Коды ошибок. В стандарте SQL2 определены стандартные коды ошибок, которые возвращают операторы SQL при возникновении ошибок.

Типы данных. В стандарте SQL2 упомянуты многие стандартные типы данных (например, символьные строки переменной длины, дата и время, а также денежные единицы), однако отсутствуют «новые» типы данных, такие как графические и мультимедийные объекты.

Системные таблицы. В стандарте SQL-89 умалчивается о системных таблицах, в которых содержится информация о структуре самой базы данных. Поэтому каждый поставщик создавал собственные системные таблицы, и их структура отличается даже в четырех реа-

лизациях SQL компании IBM. В SQL2 системные таблицы стандартизованы.

Интерактивный SQL. В стандарте SQL-89 определен только *программный* SQL, используемый прикладной программой, но не интерактивный SQL. Например, оператор SELECT, предназначенный для выполнения запросов к базе данных в интерактивном режиме, в стандарте отсутствует.

Программный интерфейс. В стандарте SQL2 определен интерфейс встроенного SQL для некоторых языков программирования, но не интерфейс вызова функций.

Динамический SQL. В стандарте SQL-89 не описаны элементы SQL, необходимые для разработки приложений общего назначения, таких как генераторы отчетов и программы создания и выполнения запросов. Однако эти элементы, известные под названием *динамический SQL,* имеются почти во всех СУБД и в различных системах значительно различаются. В стандарт SQL2 входит раздел динамического SQL.

Семантические отличия. Поскольку некоторые элементы определены в стандартах как зависящие от реализации, может возникнуть ситуация, когда в результате выполнения одного и того же запроса в двух совместимых СУБД будут получены два различных набора результатов. Отличия результатов обусловлены различиями в обработке значений NULL, разными агрегатными функциями и несовпадением процедур удаления повторяющихся строк.

Последовательность сравнения. Стандарт SQL2 позволяет программе или пользователю указывать требуемую последовательность сортировки результатов запроса.

Структура базы данных. В стандарте SQL-89 определен язык, операторы которого используются уже после того, как база данных была открыта и подготовлена к работе. Детали именования баз данных и первоначального подключения к ним в разных реализациях сильно различаются или несовместимы. Стандарт SQL2 в некоторой степени унифицирует этот процесс, хотя и не может полностью ликвидировать все различия.

Основными направлениями дальнейшего развития SQL2 (и принятия стандарта SQL:1999 — SQL3) явились:

- стандартизация интерфейсов вызова функций;
- стандартизация поддержки рекурсивных запросов;
- стандартизация хранимых процедур;
- добавление объектно ориентированных возможностей.

Следующие версии стандартов (SQL:2003, SQL:2006, SQL:2008) вводят расширения для работы с XML-документами и базами данных OLAP.

В главе 9 описаны синтаксис, семантика и приведены примеры основных инструкций языка SQL.

8.4. QBE и создание запросов на выборку данных

Язык Query by Example (запрос по образцу) представляет собой пример декларативного языка манипулирования данными. Основное назначение языка — обеспечить формулировку запроса на управление данными «по образцу».

Для реляционных БД язык QBE включает набор средств конструирования запроса в виде структуры таблицы-результата. Столбцами таблицы-результата являются атрибуты (или результаты вычисления функций, аргументами которых являются атрибуты) одной или нескольких таблиц БД, участвующие в совокупном критерии отбора данных и/или отображающиеся как результат запроса. Строки результата-таблицы задают отдельно для каждого атрибута простое условие отбора, сортировки и вывода атрибута. При этом составной критерий отбора получается путем соединения простых условий предопределенными логическими операциями (AND или OR).

Рассмотрим язык QBE СУБД Microsoft Access на примере запроса на выборку данных. Такие запросы предназначены для извлечения данных из одной или нескольких таблиц и отображения полученных результатов в виде таблицы, допускающей (с некоторыми ограничениями) модификацию содержащихся в ней записей. Запросы на выборку могут использовать группировку записей и вывод результатов применения агрегирующих функций (количества записей, сумм, средних значений и т. п.).

Для формирования запроса на выборку данных в СУБД Microsoft Access служит инструмент, называемый конструктором запроса. При создании запроса в режиме конструктора открывается окно «Запрос на выборку» и одновременно на экран выводится диалоговое окно «Добавление таблицы», содержащее сгруппированные по страницам списки объектов БД, которые могут принимать участие в построении запроса (в СУБД Microsoft Access — таблицы и ранее созданные запросы). В этом окне необходимо указать и выбрать в окно запроса все

объекты, которые требуются для формулировки условия поиска и вывода результата. На рис. 8.3 представлен пример конструктора запроса на выборку данных для БД «Сессия».

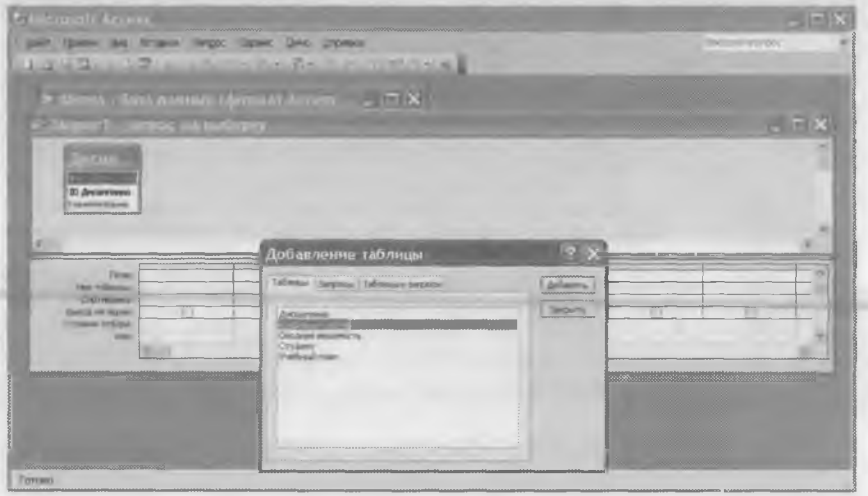


Рис. 8.3. Конструктор запроса на выборку данных

Следующий этап формирования запроса — задание условий отбора, т. е. ограничений на результаты выполнения запроса с целью выборки только тех записей данных, которые представляют интерес для пользователя. Например, для извлечения данных о студентах, сдавших экзамены на отличные оценки, средствами QBE может быть подготовлен запрос, показанный на рис. 8.4. Результаты, полученные после выполнения этого запроса, будут отображены в таблице данных, содержащей отмеченные для вывода столбцы.

Условие отбора можно усложнить, вводя дополнительные критерии в том же столбце или в других столбцах таблицы. Для формирования составного условия отбора используются строки таблицы QBE, начиная со строки «Условие отбора» и ниже. Формирование составного условия отбора осуществляется в соответствии со следующими правилами: если простые условия отбора будут помещены в одну и ту же строку, то для их соединения используется логическая операция AND («и»); если простые условия будут помещены в разные строки таблицы QBE, то для их соединения используется логическая операция OR («или»).

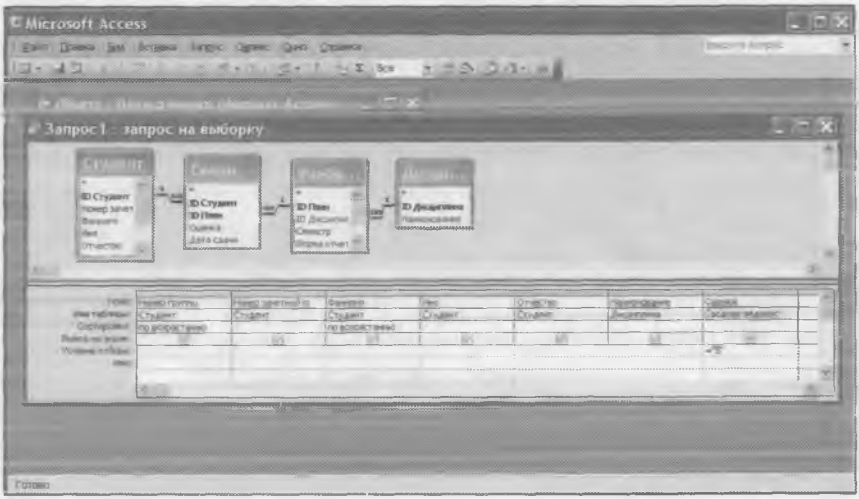


Рис. 8.4. Запрос на выборку данных

При создании запроса в таблице формирования запроса QBE СУБД Microsoft Access неявно генерирует для него эквивалентную инструкцию SQL. Просмотреть и отредактировать эту инструкцию можно в окне SQL. Текст оператора SQL, эквивалентного запросу, изображенному на рис. 8.4, показан на рис. 8.5.

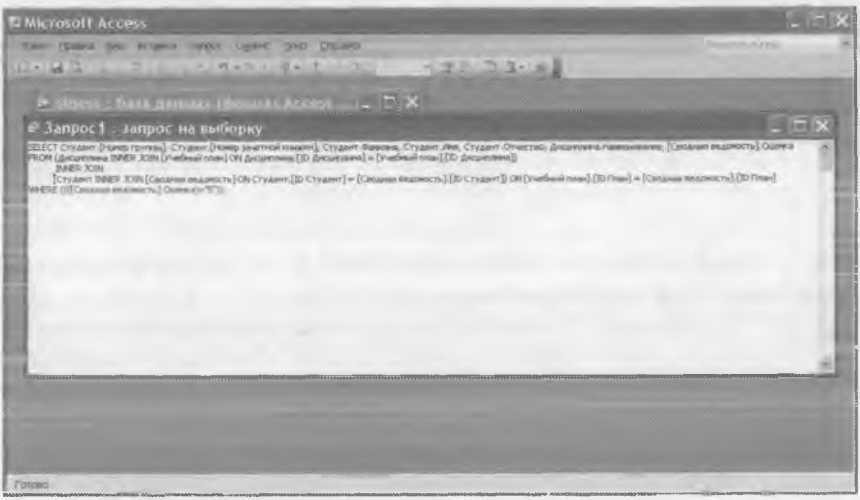


Рис. 8.5. Текст QBE-запроса на языке SQL

8.5. Языки 4GL

4GL — это аббревиатура английского варианта термина *язык четвертого поколения* (Fourth-Generation Language).

В отличие от языков третьего поколения, которые в основе своей являются процедурными, языки 4GL выступают как непроцедурные, поскольку служат для определения того, *что* должно быть сделано, оставляя «за кадром» способ достижения желаемого результата. Реализация языков четвертого поколения должна быть в значительной степени основана на использовании компонентов высокого уровня, которые называют «инструментами четвертого поколения». Такие инструменты служат строительными модулями, из которых необходимо выбрать те, которые отвечают решению поставленной задачи, и указать для них параметры. Далее осуществляется автоматическая генерация кода приложения, которое выполняет определенную функцию. Тем самым языки четвертого поколения могут существенно повысить производительность работы, но произойдет это за счет ограничения типов задач, которые можно будет решать с их помощью.

Примерами языков четвертого поколения могут служить выше упоминавшиеся SQL и QBE. Также к языкам 4GL можно отнести генераторы форм, генераторы отчетов, генераторы приложений.

Генератор форм представляет собой интерактивный графический инструментарий, предназначенный для создания шаблонов экранных форм для ввода и отображения данных. В состав инструментов входят элементы управления (поля ввода, кнопки, переключатели и т. п.), цветовая палитра, шрифты. Генератор форм позволяет определить не только внешний вид экранной формы (цвета элементов экрана, шрифтовые выделения надписей) и место расположения на экране, но и ее содержимое: расположить поля ввода и отображения элементов данных, задать правила формально-логического контроля, оформить и связать с элементами управления (кнопками) макросы.

На рис. 8.6 представлена форма ввода данных «Информация о студенте» для заполнения таблицы «Студент» БД «Сессия» в режиме конструктора. На рис. 8.7 эта же форма представлена в режиме ввода данных.

Генератор отчетов служит для формирования отчетов на основе хранимой в базе данных информации. Отчет формируется, как правило, на основе одного или нескольких фиксированных запросов к БД и выводится по мере изменения параметров запросов. Генератор отчетов предоставляет средства создания таких запросов к базе дан-

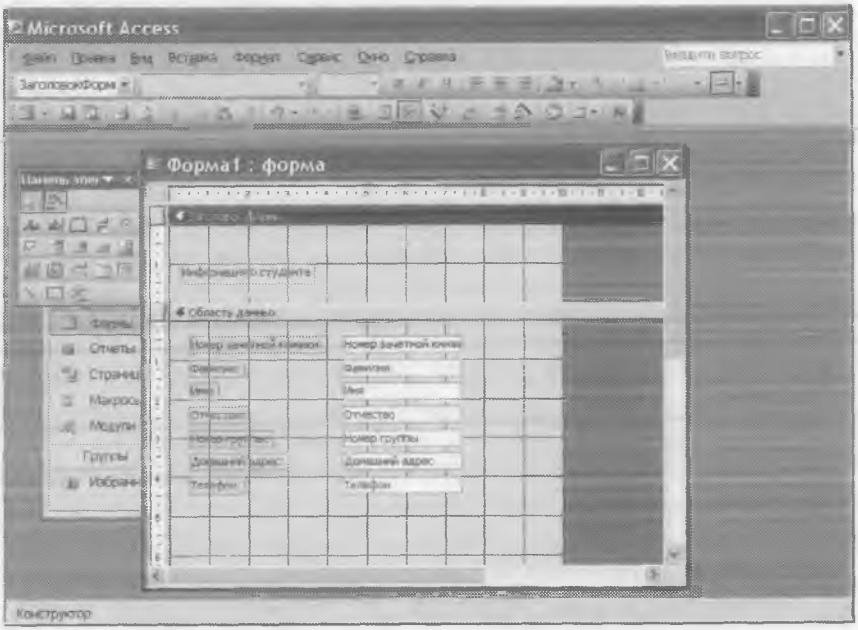


Рис. 8.6. Форма ввода в режиме «Конструктор»

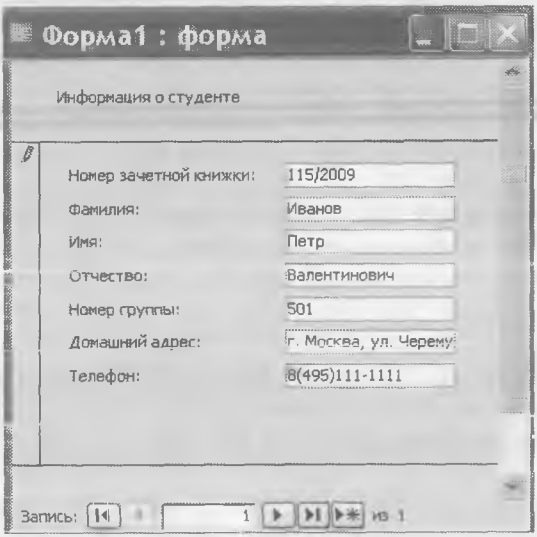


Рис. 8.7. Форма ввода в режиме ввода данных

ных и извлечения данных, используемых для представления в отчете. Кроме того, генераторы отчетов предоставляют гибкие средства управления внешним видом отчета и выводом его на устройство отображения.

Различают два основных типа генераторов отчетов: *языковой* и *визуальный*. В первом случае для определения нужных для отчета данных и внешнего вида документа следует ввести соответствующие команды на некотором подъязыке. Во втором случае для этих целей используется интерактивный графический инструментарий, подобный генератору форм.

Контрольные вопросы

1. Охарактеризуйте назначение языка определения данных.
2. Охарактеризуйте назначение язык манипулирования данными.
3. Перечислите типы языков манипулирования данными.
4. К какому типу языков можно отнести SQL?
5. К какому типу языков можно отнести QBE?
6. Охарактеризуйте возможности QBE по выборке данных.
7. Приведите примеры языков 4GL.
8. Охарактеризуйте основное назначение генераторов форм и генераторов отчетов.

Глава 9
ВВЕДЕНИЕ В SQL

Все примеры использования языка SQL, рассматриваемые в настоящей главе, построены на основе учебной базы данных «Сессия», проект которой представлен в главе 7.

9.1. Основные понятия и компоненты[1]

9.1.1. Инструкции и имена

SQL представлен множеством инструкций, каждая из которых предписывает СУБД выполнить определенное действие: создать таблицу, извлечь данные, добавить в таблицу новые данные и т. п. Инструкция SQL начинается с *команды* — ключевого слова, описывающего действие, выполняемое инструкцией. Типичными являются команды CREATE (создать), INSERT (добавить), SELECT (выбрать), DELETE (удалить). Следом за командой указывается одно или несколько *предложений*. Предложение описывает данные, с которыми должна работать инструкция, или уточняет действие, выполняемое инструкцией. Предложения в инструкции делятся на обязательные и необязательные. Каждое предложение начинается с ключевого слова, например — WHERE (где), FROM (откуда), INTO (куда). Многие предложения в качестве параметров содержат имена таблиц или столбцов; некоторые из них могут содержать дополнительные ключевые слова, константы и выражения.

У каждого объекта в базе данных есть уникальное *имя*. Имена используются в инструкциях SQL и указывают, над каким объектом базы данных инструкция должна выполнить действие. В соответствии

[1] Синтаксис приведенных в этой главе примеров соответствует MS SQL Server 2008.

со стандартом ANSI/ISO имена в SQL могут содержать от 1 до 18 символов, начинаться с буквы и не должны включать пробелов или специальных символов пунктуации. В стандарте SQL2 максимальное число символов в имени увеличено до 128. На практике в различных СУБД поддержка именования реализована по-разному: в DB2, например, имена пользователей не могут превышать восьми символов, а имена таблиц и столбцов могут быть более длинными. В различных СУБД также существуют и различные подходы к использованию в именах специальных символов.

В инструкциях SQL могут использоваться как полные имена объектов, так и короткие. Полное имя таблицы (в отличие от короткого) содержит имя пользователя и короткое имя таблицы, разделенные точкой:

```
<Имя_пользователя>.<Имя_таблицы>
```

При этом уникальность именования таблицы сохраняется в случае, если в рамках одной базы данных разные пользователи создают таблицы с одинаковыми именами.

Полное имя столбца в свою очередь состоит из полного (или короткого) имени таблицы, которой принадлежит столбец, и короткого имени столбца, разделенных точкой:

```
<Имя_пользователя>.<Имя_таблицы>.<Имя_столбца> или
<Имя_таблицы>.<Имя_столбца>
```

В рамках одной таблицы не может быть определено двух столбцов с одинаковыми именами, но в разных таблицах это возможно. При этом в инструкциях SQL необходимо использовать полное именование столбцов.

9.1.2. Типы данных

Современные СУБД позволяют обрабатывать данные разнообразных типов, среди которых наиболее распространенными можно назвать следующие:

- целые числа;
- десятичные числа;
- числа с плавающей запятой;
- строки символов постоянной длины;
- строки символов переменной длины;

- денежные величины;
- дата и время;
- булевы величины;
- неструктурированные потоки байтов.

Рассмотрим разновидности перечисленных типов данных, реализованные в СУБД MS SQL Server 2008.

Целые числа. Представлены множеством типов, характеристики которых приведены в табл. 9.1.

Таблица 9.1. **Целые типы данных**

Тип данных	Диапазон	Объем памяти (в байтах)
bigint	-2^{63}—$2^{63}-1$	8
int	-2^{31}—$2^{31}-1$	4
smallint	-2^{15}—$2^{15}-1$	2
tinyint	0—255	1

Десятичные числа. В столбцах данного типа хранятся числа, имеющие дробную часть с фиксированным количеством знаков после запятой, например курсы валют и проценты. Десятичные числа представлены двумя функционально равнозначными типами — decimal(*p*[,*s*]) и numeric(*p*[,*s*]). Максимальное количество десятичных разрядов числа задается точностью (*p*), принимающей значения от 1 до 38. По умолчанию точность принимает значение, равное 18. Максимальное количество десятичных разрядов числа справа от десятичной запятой задается масштабом (*s*). Масштаб может быть указан только вместе с точностью и может принимать значения от 0 до *p*. По умолчанию масштаб принимает значение, равное 0.

Максимальный размер памяти, отводимой под хранение десятичных чисел, зависит от точности (табл. 9.2). При использовании максимальной точности числа могут принимать значения в диапазоне от $-10^{38}+1$ до $10^{38}-1$.

Таблица 9.2. **Десятичные числа**

Точность (в разрядах)	Объем памяти (в байтах)
1—9	5
10—19	9
20—28	13
29—38	17

Числа с плавающей запятой. Числа с плавающей запятой представляют больший диапазон действительных значений, чем десятичные числа (табл. 9.3).

Таблица 9.3. **Числа с плавающей запятой**

Тип данных	Диапазон	Объем памяти (в байтах)
float	–1,79E+308—–2,23E-308, 0 и 2,23E-308—1,79E+308	4 или 8
real	–3,40E + 38 — –1,18E — 38, 0 и 1,18E – 38—3,40E + 38	4

Строки символов постоянной длины. При хранении таких данных введенное значение дополняется до заданной длины пробелами (табл. 9.4).

Таблица 9.4. **Строки символов постоянной длины**

Тип данных	Описание	Максимальный размер (в символах)
char(*n*)	Символьные данные фиксированной длины размером в *n* байтов	8000
nchar(*n*)	Двоичные данные переменной длины в Юникоде с максимальной длиной *n* байтов	4000

Строки символов переменной длины. Столбцы этого типа позволяют хранить символьные строки, длина которых изменяется в заданном диапазоне. Для типов данных, поддерживающих хранение символьных строк в Юникоде, объем занимаемой памяти в байтах в два раза превышает число символов (табл. 9.5).

Денежные величины. Наличие отдельного типа данных для хранения денежных величин позволяет правильно форматировать их и снабжать признаком валюты перед выводом на экран или печать (табл. 9.6).

Дата и время. Поддержка особого типа данных для значений дата/время широко распространена в СУБД. Как правило, с этим типом данных связаны особые операции и процедуры обработки (табл. 9.7).

Булевы величины (bit). Столбцы такого типа данных позволяют хранить логические значения True (1) и False (0).

Таблица 9.5. **Строки символов переменной длины**

Тип данных	Описание	Максимальный размер (в символах)
varchar(*n*)	Символьные данные переменной длины с максимальной длиной *n* символов	8000
varchar(*max*)	Символьные данные переменной длины. Размер памяти для хранения — фактическая длина введенных данных плюс 2 байта. Введенные данные могут иметь длину 0 символов	$2^{31} - 1$
nvarchar(*n*)	Символьные данные переменной длины в Юникоде с максимальной длиной *n* символов	4000
nvarchar(*max*)	Символьные данные переменной длины в Юникоде. К размеру хранения символьных данных в байтах добавляется два байта. Введенные данные могут иметь длину 0 символов	$2^{30} - 1$
text	Символьные данные переменной длины	$2^{31} - 1$
ntext	Символьные данные переменной длины в Юникоде	$2^{30} - 1$

Таблица 9.6. **Денежные величины**

Тип данных	Диапазон	Объем памяти (в байтах)
money	–922 337 203 685 477,5808—922 337 203 685 477,5807	8
smallmoney	–214 748,3648—214 748,3647	4

Таблица 9.7. **Типы данных для даты и времени**

Тип данных	Диапазон даты	Диапазон времени	Объем памяти (в байтах)
date	1 января 1 года (0001-01-01) — 31 декабря 9999 года (9999-12-31)		3
time		00:00:00 — 23:59:59.9999999	5
smalldatetime	1 января 1900 года (1900-01-01) — 6 июня 2079 года (2079-06-06)	00:00:00—23:59:59	4

Окончание табл. 9.7

datetime	1 января 1753 года (1753-01-01) — 31 декабря 9999 года (9999-12-31)	00:00:00—23:59:59.997	8
datetime2	1 января 1 года (0001-01-01) — 31 декабря 9999 года (9999-12-31)	00:00:00— 23:59:59.9999999	6—8

Неструктурированные потоки байтов. (BINARY, VARBINARY, IMAGE). Современные СУБД позволяют хранить и извлекать неструктурированные потоки байтов переменной длины. Такой тип данных обычно используется для хранения графических и видеоизображений, исполняемых файлов и других неструктурированных данных (табл. 9.8).

Таблица 9.8. **Неструктурированные потоки байтов**

Тип данных	Описание	Максимальный размер (в байтах)
binary(*n*)	Двоичные данные фиксированной длины размером в *n* байтов	8000
varbinary(*n*)	Двоичные данные переменной длины с максимальной длиной *n* байтов	8000
varbinary(*max*)	Двоичные данные переменной длины. Размер памяти для хранения — фактическая длина введенных данных плюс 2 байта. Введенные данные могут иметь размер 0 символов	$2^{31} - 1$
image	Двоичные данные переменной длины	$2^{31} - 1$

В СУБД MS SQL Server 2008 реализованы также следующие типы данных:

- sql_variant — для хранения значения любых типов данных, поддерживаемых SQL Server;
- geography, geometry — для хранения пространственных данных;
- xml — для хранения XML-данных.

9.1.3. Встроенные функции

Язык SQL содержит так называемые встроенные функции, которые реализуют некоторые наиболее распространенные алгоритмы. Основной особенностью этих функций является возможность их использования при построении выражений.

Встроенные функции, доступные при работе с SQL, можно условно разделить на следующие группы:

- математические функции;
- строковые функции;
- функции для работы с величинами типа дата-время;
- функции конфигурирования;
- системные функции;
- функции системы безопасности;
- функции управления метаданными;
- статистические функции.

В табл. 9.9 приведены наиболее часто используемые функции первых трех групп.

Таблица 9.9. **Некоторые встроенные функции SQL**

Функция	Назначение
ABS(число)	Вычисляет абсолютную величину числа
ISNUMERIC(выражение)	Определяет, имеет ли выражение числовой тип данных
SIGN(число)	Определяет знак числа
RAND(целое число)	Вычисляет случайное число с плавающей запятой в интервале от 0 до 1
ROUND(число, точность)	Выполняет округление числа с указанной точностью
POWER(число, степень)	Возводит число в степень
SQRT(число)	Извлекает квадратный корень из числа
SIN(угол)	Вычисляет синус угла, указанного в радианах
COS(угол)	Вычисляет косинус угла, указанного в радианах
EXP(число)	Вычисляет экспоненту числа
LOG(число)	Вычисляет натуральный логарифм числа
LEN(строка)	Вычисляет длину строки в символах

Окончание табл. 9.9

Функция	Назначение
LTRIM(строка)	Удаляет пробелы в начале строки
RTRIM(строка)	Удаляет пробелы в конце строки
LEFT(строка, количество)	Возвращает указанное количество символов строки, начиная с самого левого символа
RIGHT(строка, количество)	Возвращает указанное количество символов строки, начиная с самого правого символа
LOWER(строка)	Приводит символы строки к нижнему регистру
UPPER(строка)	Приводит символы строки к верхнему регистру
STR(число)	Выполняет конвертирование числового значения в символьный формат
SUBSTRING(строка, индекс, длина)	Возвращает для строки подстроку заданной длины, начиная с символа заданного индекса
GETDATE()	Возвращает текущую системную дату
ISDATE(строка)	Проверяет строку на соответствие одному из форматов даты и времени
DAY(дата)	Возвращает число указанной даты
MONTH(дата)	Возвращает месяц указанной даты
YEAR(дата)	Возвращает год указанной даты
DATEADD(тип, число, дата)	Прибавляет к дате указанное число единиц заданного типа (год, месяц, день, час и т. п.)

9.1.4. Значения NULL

При заполнении таблиц базы данных отдельные элементы в них могут отсутствовать. Например, при заполнении таблицы «Студенты» или «Кадровый_состав» может быть не задан для некоторых строк номер телефона, тем не менее строка должна быть введена в таблицу и должна участвовать в запросах на выдачу информации.

SQL поддерживает обработку не определенных (не заданных) данных с помощью использования так называемого отсутствующего значения (NULL). Это значение показывает, что в конкретной строке конкретный элемент данных отсутствует. При этом NULL не является значением данных и в связи с этим не имеет определенного типа.

Это всего лишь признак, показывающий, что значение элемента данных не задано.

Правила обработки значений NULL в различных инструкциях и предложениях включены в синтаксис языка.

9.2. Ограничения целостности

9.2.1. Первичный ключ таблицы

Всякая таблица обычно содержит один или несколько столбцов, значение или совокупность значений которых *уникально идентифицируют* каждую строку в таблице. Этот столбец (или столбцы) называется *первичным ключом* (Primary Key, PK) таблицы.

Если в первичный ключ входит более одного столбца, значения в пределах одного столбца могут дублироваться, но любая совокупность значений всех столбцов первичного ключа при этом должна быть уникальна. Например, в таблице «Дисциплины» один столбец (*ID_Дисциплина*) определен как первичный ключ (рис. 9.1), а для таблицы «Сводная ведомость» задан составной первичный ключ — в него входят значения столбцов *ID_Студент* и *ID_Дисциплина*.

Учебный_план

Имя столбца	Тип данных	Длина	Допускает значения NULL
ID_План	int	4	Нет
ID_Дисциплина	int	4	Нет
Семестр	tinyint	1	Нет
Форма_отчетности	char(1)	1	Нет
Количество_часов	smallint	2	Да
Табельный_номер	int	4	Да

Рис. 9.1. Первичный ключ таблицы «Учебный_план»

Таблица может иметь *только один* первичный ключ, причем никакой столбец, входящий в первичный ключ, не может хранить значение NULL.

Еще одним назначением первичного ключа является обеспечение ссылочной целостности данных в нескольких таблицах. Естественно, это может быть реализовано только при наличии соответствующих *внешних ключей* (FOREIGN KEY) в других (дочерних) таблицах.

Если по столбцу строится первичный ключ, столбцу должен быть приписан атрибут PRIMARY KEY (ограничение целостности на уровне столбца); например, описание столбца *ID_План* для таблицы «Учебный_план» (см. рис. 9.1) может выглядеть так:

```
ID_Дисциплина INTEGER NOT NULL PRIMARY KEY
```

Первичный ключ может быть также построен с помощью отдельного предложения PRIMARY KEY (ограничение целостности на уровне таблицы) — путем включения имени (имен) ключевого столбца (столбцов) в качестве параметров. Например, первичный ключ для таблицы «Сводная_ведомость» (рис. 9.2) может быть задан следующим образом:

```
PRIMARY KEY (ID_Дисциплина, ID_Студент)
```

Сводная_ведомость

Имя столбца	Тип данных	Длина	Допускает значения NULL
ID_Студент	int	4	Нет
ID_Дисциплина	int	4	Нет
Оценка	tinyint	1	Нет
Дата_сдачи	datetime	8	Нет

Рис. 9.2. Первичный ключ таблицы «Сводная_ведомость»

9.2.2. Внешний ключ таблицы

Внешний ключ строится в дочерней (зависимой) таблице для соединения родительской (главной) и дочерних таблиц БД.

Это ограничение целостности предназначено для организации ссылочной целостности данных. Внешний ключ связывается с потенциальным первичным ключом в другой таблице. Внешний ключ при этом может ссылаться либо на столбец (или столбцы) с ограничением целостности PRIMARY KEY, либо на столбец (столбцы) с ограничением целостности UNIQUE.

Таблицу, в которой определен внешний ключ, будем называть *зависимой*, а таблицу с первичным ключом — *главной*. Ссылочная целостность данных двух таблиц обеспечивается следующим образом: в зависимую таблицу нельзя вставить строку, если внешний ключ не

имеет соответствующего значения в главной таблице, а из главной таблицы нельзя удалить строку, если значение первичного ключа используется в зависимой таблице.

Например, если строка наименования дисциплины удалена из таблицы «Дисциплины», а идентификатор этой дисциплины (*ID_Дисциплина*) используется в таблице «Учебный_план», то относительная целостность между этими двумя таблицами будет нарушена — строки таблицы «Учебный_план» с удаленным идентификатором останутся «осиротевшими». Ограничение FOREIGN KEY предотвращает возникновение подобных ситуаций — удаление строки первичного ключа не состоится.

Столбцы внешнего ключа (в отличие от столбцов первичного ключа) могут содержать значения типа NULL, однако при этом проверка на ограничение FOREIGN KEY будет пропускаться. Задать внешний ключ можно как при создании, так и при изменении таблиц.

Синтаксис определения внешнего ключа следующий:

```
FOREIGN KEY (<список столбцов внешнего ключа>)
REFERENCES <имя родительской таблицы>
[[<список столбцов родительской таблицы>]
[ON DELETE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
[ON UPDATE {NO ACTION | CASCADE | SET DEFAULT| SET NULL}]
```

Список столбцов внешнего ключа определяет столбцы дочерней таблицы, по которым строится внешний ключ.

Имя родительской таблицы определяет таблицу, в которой описан первичный ключ (или столбец с атрибутом UNIQUE). На этот ключ (столбец) должен ссылаться внешний ключ дочерней таблицы для обеспечения ссылочной целостности.

Список столбцов родительской таблицы, определяющий ссылочную целостность, необязателен при ссылке на первичный ключ родительской таблицы. При ссылке в родительской таблице на столбец с атрибутом UNIQUE этот список лучше привести.

Параметры ON DELETE, ON UPDATE задают способы изменения подчиненных записей дочерней таблицы при удалении (ON DELETE) или изменении (ON UPDATE) поля связи в записи родительской таблицы. Перечислим эти способы:

- NO ACTION — запрещает удаление/изменение родительской записи при наличии подчиненных записей в дочерней таблице;

- CASCADE — при удалении записи родительской таблицы (используется совместно с ON DELETE) происходит удаление всех подчиненных записей в дочерней таблице; при изменении поля связи в записи родительской таблицы (используется совместно с ON UPDATE) происходит изменение на то же значение поля внешнего ключа у всех подчиненных записей в дочерней таблице;
- SET DEFAULT — в поле внешнего ключа записей дочерней таблицы заносится значение этого поля по умолчанию, указанное при определении поля (параметр DEFAULT);
- SET NULL — в поле внешнего ключа записей дочерней таблицы заносится значение NULL.

Установим связь между таблицами «Студенты», «Учебный_план» и «Сводная_ведомость»:

```
ALTER TABLE Сводная_ведомость
ADD FOREIGN KEY (ID_План)
  REFERENCES Учебный_план

ALTER TABLE Сводная_ведомость
ADD FOREIGN KEY (ID_Студент)
  REFERENCES Студенты
```

Рис. 9.3. Связь внешнего и первичного ключей

Хотя в рассмотренном примере имена столбцов первичного и внешнего ключей в обеих таблицах совпадают, это не является обязательным. Первичный ключ может быть определен для столбца с одним именем, в то время как столбец, на который наложено ограничение FOREIGN KEY, может иметь совершенно другое имя. Однако лучше давать таким столбцам идентичные названия, чтобы показать связь между ними (рис. 9.3).

9.2.3. Определение уникального столбца

Ограничение целостности UNIQUE предназначено для того, чтобы обеспечить уникальность значений в столбце (или нескольких столбцах). Если столбцу приписан атрибут UNIQUE, это означает, что в столбце не могут содержаться два одинаковых значения.

Для ограничения целостности PRIMARY KEY автоматически гарантируется уникальность значений. Однако в каждой таблице можно определить всего один первичный ключ. Если же необходимо дополнительно обеспечить уникальность значений еще в одном или более столбцах помимо первичного ключа, то нужно использовать ограничение целостности UNIQUE.

Ограничение целостности UNIQUE, в отличие от PRIMARY KEY, допускает существование значения NULL. При этом к значению NULL также предъявляется требование уникальности, поэтому в столбце с ограничением целостности UNIQUE допускается существование лишь единственного значения NULL.

Таким образом, ограничение UNIQUE используется в том случае, когда столбец не входит в состав первичного ключа, но, тем не менее, его значение всегда должно быть уникальным. Например, для таблицы «Дисциплины» первичный ключ строится по номеру дисциплины *ID_Дисциплина*, введенному для сокращения объема первичного ключа и времени поиска по нему (объем ключа по столбцу типа INTEGER много меньше объема ключа по символьному полю). Однако и название дисциплины (столбец *Наименование*) должно быть уникальным, для чего ему приписан атрибут UNIQUE:

```
CREATE TABLE Дисциплины
(ID_Дисциплина   INTEGER NOT NULL PRIMARY KEY,
Наименование     VARCHAR(20) NOT NULL UNIQUE)
```

Уникальность может быть определена и на уровне таблицы:

```
CREATE TABLE Дисциплины
(ID_Дисциплина  INTEGER NOT NULL,
Наименование    VARCHAR(20) NOT NULL,
PRIMARY KEY     (ID_Дисциплина),
UNIQUE          (Наименование))
```

9.2.4. Определение проверочных ограничений

Ограничение целостности CHECK задает диапазон возможных значений для столбца. Например, если в столбце хранится процентное значение, то необходимо гарантировать, что оно будет лежать в пределах от 0 до 100. Для этого можно использовать тип данных, допускающий хранение целых значений в диапазоне от 0 до 255, совместно с ограничением целостности CHECK, которое будет обеспечивать соответствующую проверку значений.

Преимуществом ограничения целостности CHECK является возможность определения для одного столбца множества правил контроля значений.

В основе ограничения целостности CHECK лежит проверка логического выражения, которое возвращает значение TRUE (истина) либо значение FALSE (ложь). Если возвращается значение TRUE, то ограничение целостности выполняется и операция изменения или вставки данных разрешается. Когда же возвращается значение FALSE, то операция изменения или вставки данных отменяется.

Например, для обеспечения правильности задания значения для столбца *Семестр* в таблице «Учебный_план» (оно должно находиться в диапазоне от 1 до 10) можно использовать следующее логическое выражение:

```
((Семестр >= 1) OR (Семестр <= 10)))
```

Ограничение целостности при этом может быть задано на уровне столбца:

```
Семестр INTEGER NOT NULL CHECK ((Семестр >= 1) OR
(Семестр <= 10)))
```

Или на уровне таблицы:

```
CHECK ((Семестр >= 1) OR (Семестр <= 10)))
```

Как уже было сказано, допускается применение нескольких ограничений CHECK к одному и тому же столбцу. В этом случае они будут применены в той последовательности, в какой они указаны в инструкции.

9.2.5. Определение значения по умолчанию

При вводе записи (строки) в таблицу каждый столбец должен содержать какое-либо значение. Если значение для столбца не указано, то столбец заполняется значениями NULL (конечно, если для него разрешено хранение значений NULL). Однако это нежелательно. Наилучшим решением в подобных ситуациях может быть определение для столбца *значений по умолчанию*. Например, часто ноль определяется как значение по умолчанию для числовых столбцов, а «n/a» (не определено) — как значение по умолчанию для символьных столбцов. Таким образом, определение для столбца значения по умолчанию гарантирует автоматическую подстановку этого значения, если при вставке новых строк значение для столбца не указано.

Использование значений по умолчанию довольно удобно, поскольку позволяет ускорить процесс ввода информации. Значительно расширяет область применения значений по умолчанию возможность вызова встроенных функций. Например, если в столбце необходимо указать дату поступления на работу, то по умолчанию можно воспользоваться функцией GETDATE(). В этом случае, если не указана другая дата, при вводе строки в столбец дат поступления на работу будет помещаться текущая дата.

9.3. Управление таблицами

9.3.1. Команда создания таблицы — CREATE TABLE

Создание таблицы выполняется при помощи команды CREATE TABLE. Обобщенный синтаксис команды следующий:

```
CREATE TABLE имя_таблицы
({<определение_столбца>|<определение_ограничения_
таблицы>}[,...,{<определение_столбца>|<определение_
ограничения_таблицы >}])
```

То есть после задания имени таблицы через запятую в круглых скобках должны быть перечислены все предложения, определяющие отдельные элементы таблицы — столбцы или ограничения целостности:

имя_таблицы — идентификатор создаваемой таблицы, который в общем случае строится из имени базы данных, имени владельца таблицы и имени самой таблицы. При этом комбинация имени таблицы и ее владельца должна быть уникальной в пределах базы данных. Если таблица создается не в текущей базе данных, в ее идентификатор необходимо включить имя базы данных;

определение_столбца — задание имени, типа данных и параметров отдельного столбца таблицы. Названия столбцов должны соответствовать правилам для идентификаторов и быть уникальными в пределах таблицы;

определение_ограничения_таблицы — задание некоторого ограничения целостности на уровне таблицы.

Описание столбцов

Как видно из синтаксиса команды CREATE TABLE, для каждого столбца указывается предложение <определение_столбца>, с помощью которого и задаются свойства столбца. Предложение имеет следующий синтаксис:

```
<Имя_столбца> <тип_данных>
[<ограничение_столбца> ] [,...,<ограничение_столбца>]
```

Рассмотрим назначение и использование параметров.

Имя_столбца — идентификатор, задающий имя столбца таблицы;

тип_данных — задает тип данных столбца. Если при определении столбца явно не указано ограничение на хранение значений NULL, то будут использованы свойства типа данных, т. е. если выбранный тип данных позволяет хранить значения NULL, то и в столбце можно будет хранить значения NULL. Если же при определении столбца в команде CREATE TABLE явно будет разрешено или запрещено хранение значений NULL, то свойства типа данных будут перекрыты установленным на уровне столбца ограничением. Например, если тип данных позволяет хранить значения NULL, а на уровне столбца будет установлен запрет, то попытка вставки значения NULL в столбец закончится ошибкой;

ограничение_столбца — с помощью этого предложения указываются ограничения, которые будут определены для столбца. Синтаксис предложения следующий:

```
<ограничение_столбца>::=[ CONSTRAINT <имя_ограничения > ]
{[ DEFAULT <выражение>]
| [ NULL | NOT NULL ]
| [ PRIMARY KEY | UNIQUE ]
| [FOREIGN KEY
REFERENCES <имя_главной_таблицы>[(<имя_столбца> [,...,n])]
[ ON DELETE { CASCADE | NO ACTION } ]
[ ON UPDATE { CASCADE | NO ACTION } ]
]
| [CHECK (<логическое_выражение>)]
}
```

Рассмотрим назначение параметров.

CONSTRAINT — необязательное ключевое слово, после которого указывается название ограничения на значения столбца (имя_ограничения). Имена ограничений должны быть уникальны в пределах базы данных.

DEFAULT — задает значение по умолчанию для столбца. Это значение будет использовано при вставке строки, если для столбца явно не указано никакое значение.

NULL | NOT NULL — ключевые слова, разрешающие (NULL) или запрещающие (NOT NULL) хранение в столбце значений NULL. Если для столбца не задано значение по умолчанию, то при вставке строки с неизвестным значением для столбца будет предприниматься попытка вставки в столбец значения NULL. Если при этом для столбца указано ограничение NOT NULL, то попытка вставки строки будет отклонена, и пользователь получит соответствующее сообщение об ошибке.

PRIMARY KEY — определение первичного ключа на уровне одного столбца (т. е. первичный ключ будет состоять только из значений одного столбца). Если необходимо сформировать первичный ключ на базе двух и более столбцов, то такое ограничение целостности должно быть задано на уровне таблицы. При этом следует помнить, что для каждой таблицы может быть создан только один первичный ключ.

UNIQUE — указание на создание для столбца ограничения целостности UNIQUE, что позволит гарантировать уникальность каждого

отдельного значения в столбце в пределах этого столбца. В таблице может быть создано несколько ограничений целостности UNIQUE.

FOREIGN KEY ... REFERENCES — указание на то, что столбец будет служить внешним ключом для таблицы, имя которой задается с помощью параметра <имя_главной_таблицы>.

(*имя_столбца* [,...,n]) — столбец или список перечисленных через запятую столбцов главной таблицы, входящих в ограничение FOREIGN KEY. При этом столбцы, входящие во внешний ключ, могут ссылаться только на столбцы первичного ключа или столбцы с ограничением UNIQUE таблицы.

ON DELETE {CASCADE | NO ACTION} — эти ключевые слова определяют действия, предпринимаемые при удалении строки из главной таблицы. Если указано ключевое слово CASCADE, то при удалении строки из главной (родительской) таблицы строка в зависимой таблице также будет удалена. При указании ключевого слова NO ACTION в подобном случае будет выдана ошибка. Значением по умолчанию является вариант NO ACTION.

ON UPDATE {CASCADE | NO ACTION} — эти ключевые слова определяют действия, предпринимаемые при модификации строки главной таблицы. Если указано ключевое слово CASCADE, то при модификации строки из главной (родительской) таблицы строка в зависимой таблице также будет модифицирована. При использовании ключевого слова NO ACTION в подобном случае будет выдана ошибка. Значением по умолчанию является вариант NO ACTION.

CHECK — ограничение целостности, инициирующее контроль вводимых в столбец (или столбцы) значений;

логическое_выражение — логическое выражение, используемое для ограничения CHECK.

Ограничения на уровне таблицы

Синтаксис команды CREATE TABLE предусматривает использование предложения <ограничение_таблицы>, с помощью которого определяются ограничения целостности на уровне таблицы. Синтаксис предложения следующий:

```
<ограничение_таблицы> ::= [ CONSTRAINT <имя_ограничения>]
{ [ { PRIMARY KEY | UNIQUE }
{(<имя_колонки> [ASC | DESC] [,...,n] )}]
| FOREIGN KEY
[ ( <имя_колонки>[,..., n ] ) ]
```

```
REFERENCES <внешняя_таблица> [ (<имя_колонки_внешней_таблицы>
[, ..., n ] ) ]
[ ON DELETE { CASCADE | NO ACTION } ]
[ ON UPDATE { CASCADE | NO ACTION } ]
| CHECK (<логическое_выражение> )
}
```

Назначение параметров совпадает с назначением аналогичных параметров предложения <ограничение_столбца>. Тем не менее в предложении <ограничение_таблицы> имеются некоторые новые параметры:

имя_колонки — столбец (или список столбцов), на которые необходимо наложить какие-либо ограничения целостности;

[ASC | DESC] — метод упорядочивания данных в индексе. Индекс создается при указании ключевых слов PRIMARY KEY, UNIQUE. При указании значения ASC данные в индексе будут упорядочены по возрастанию, при указании значения DESC — по убыванию. По умолчанию используется значение ASC.

Примеры создания таблиц

В качестве примера рассмотрим инструкции создания таблиц базы данных «Сессия».

Таблица «Студенты» состоит из следующих столбцов:

ID_Студент — тип данных INTEGER, уникальный ключ;
Фамилия — тип данных CHAR, длина 30;
Имя — тип данных CHAR, длина 15;
Отчество — тип данных CHAR, длина 20;
Номер_группы — тип данных CHAR, длина 6;
Адрес — тип данных CHAR, длина 30;
Телефон — тип данных CHAR, длина 8.

Создание таблицы выполнялось с помощью следующей команды:

```
CREATE TABLE Студенты
(ID_Студент   INTEGER NOT NULL,
Фамилия       CHAR(30) NOT NULL,
Имя           CHAR(15) NOT NULL,
Отчество      CHAR(20) NOT NULL,
Номер_группы INTEGER NOT NULL,
Адрес         CHAR(30),
Телефон       CHAR(8),
PRIMARY KEY   (ID_Студент))
```

На все столбцы таблицы, кроме столбцов *Адрес* и *Телефон,* наложены ограничения NOT NULL, запрещающие ввод строки при неопределенном значении столбца.

Для создания таблицы «Дисциплины» была использована команда:

```
CREATE TABLE Дисциплины
(ID_Дисциплина  INTEGER NOT NULL,
Наименование    VARCHAR(40) NOT NULL,
PRIMARY KEY     (ID_Дисциплина),
UNIQUE          (Наименование))
```

Таблица содержит два столбца (*ID_Дисциплина*, *Наименование*).

На столбцы *ID_Дисциплина*, *Наименование* наложены ограничения NOT NULL, запрещающие ввод строки при неопределенном значении столбца.

Столбец *ID_Дисциплина* объявлен первичным ключом, а на значения, вводимые в столбец *Наименование,* наложено условие уникальности.

Таблица «Учебный_план» включает в себя следующие столбцы:

ID_План — тип данных INTEGER, столбец уникального ключа;
ID_Дисциплина — тип данных INTEGER;
Семестр — тип данных INTEGER;
Количество_часов — тип данных INTEGER;
ID_Преподаватель — тип данных INTEGER.

Создание таблицы выполнялось с помощью следующей команды:

```
CREATE TABLE Учебный_план
(ID_План          INTEGER NOT NULL,
ID_Дисциплина     INTEGER NOT NULL,
Семестр           INTEGER NOT NULL,
Количество_часов  INTEGER,
ID_Преподаватель  INTEGER,
PRIMARY KEY       (ID_План),
CHECK             ((Семестр >= 1) OR (Семестр <= 10)))
```

Для значений столбца *Семестр* сформулировано логическое выражение, разрешающее вводить только значения от 1 до 10.

Таблица «Сводная_ведомость» состоит из следующих столбцов:

ID_Студент — тип данных INTEGER, столбец уникального ключа;
ID_План — тип данных INTEGER, столбец уникального ключа;

Оценка — тип данных INTEGER;
Дата_сдачи — тип данных DATETIME;
ID_Преподаватель — тип данных INTEGER.

Создание таблицы выполнялось с помощью следующей команды:

```
CREATE TABLE Сводная_ведомость
(ID_Студент   INTEGER NOT NULL,
ID_План       INTEGER NOT NULL,
Оценка        INTEGER NOT NULL,
Дата_сдачи    DATETIME NOT NULL,
PRIMARY KEY   (ID_Студент, ID_Дисциплина),
CHECK         ((Оценка >= 0) OR (Оценка <= 5)))
```

На все столбцы таблицы наложены ограничения NOT NULL, запрещающие ввод строки при неопределенном значении столбца.

Для значений столбца *Оценка* сформулировано логическое выражение, разрешающее вводить только значения от 0 до 5: 0 — незачет, 1 — зачет, 2 — неудовлетворительно, 3 — удовлетворительно, 4 — хорошо, 5 — отлично.

И, наконец, перечислим столбцы таблицы «Кадровый_состав»:

ID_Преподаватель — тип данных INTEGER, уникальный ключ;
Фамилия — тип данных CHAR, длина 30;
Имя — тип данных CHAR, длина 15;
Отчество — тип данных CHAR, длина 20;
Должность — тип данных CHAR, длина 20;
Кафедра — тип данных CHAR, длина 3;
Адрес — тип данных CHAR, длина 30;
Телефон — тип данных CHAR, длина 8.

Создание таблицы выполнялось с помощью следующей команды:

```
CREATE TABLE Кадровый_состав
(ID_ Преподаватель INTEGER NOT NULL,
Фамилия            CHAR(30) NOT NULL,
Имя                CHAR(15) NOT NULL,
Отчество           CHAR(20) NOT NULL,
Должность          CHAR(20) NOT NULL,
Кафедра            CHAR(3) NOT NULL,
Адрес              CHAR(30),
Телефон            CHAR(8),
PRIMARY KEY        (ID_Преподаватель))
```

На все столбцы таблицы, кроме столбцов *Адрес* и *Телефон,* наложены ограничения NOT NULL, запрещающие ввод строки при неопределенном значении столбца.

Для таблиц «Учебный_план» и «Сводная_ведомость» должны быть построены внешние ключи, связывающие таблицы базы данных «Сессия»:

- FK_Дисциплина — внешний ключ, связывающий таблицы «Учебный_план» и «Дисциплины» по столбцу *ID_Дисциплина*;
- FK_Кадровый_состав — внешний ключ, связывающий таблицы «Учебный_план» и «Кадровый_состав» по столбцу *ID_Преподаватель;*
- FK_Студент — внешний ключ, связывающий таблицы «Сводная_ведомость» и «Студенты» по столбцу *ID_Студент;*
- FK_План — внешний ключ, связывающий таблицы «Сводная_ведомость» и «Учебный_план» по столбцу *ID_План.*

Добавление внешних ключей в таблицы будет описано при рассмотрении возможностей команды ALTER TABLE.

9.3.2. Изменение структуры таблицы — команда ALTER TABLE

Как бы тщательно ни планировалась структура таблицы, иногда возникает необходимость внести в нее некоторые изменения. Предположим, что в уже сформированную таблицу «Преподаватели» необходимо добавить номер домашнего телефона и домашний адрес. Эту операцию можно выполнять различными путями. Например, можно удалить таблицу со старой структурой и создать вместо нее новую таблицу с нужной структурой. Недостатком этого метода является то, что необходимо будет куда-то скопировать имеющиеся в таблице данные и переписать их в новую таблицу после ее создания.

Специальная команда ALTER TABLE предназначена для *модификации* структуры таблицы. С ее помощью можно изменять свойства существующих столбцов, удалять или добавлять в таблицу столбцы, а также управлять ограничениями целостности как на уровне столбца, так и на уровне таблицы, т. е. выполнять следующие функции:

- добавить в таблицу определение нового столбца;
- удалить столбец из таблицы;
- изменить значение по умолчанию для какого-либо столбца;
- добавить или удалить первичный ключ таблицы;

- добавить или удалить внешний ключ таблицы;
- добавить или удалить условие уникальности;
- добавить или удалить условие на значение.

Рассмотрим обобщенный синтаксис команды ALTER TABLE:

```
ALTER TABLE <имя_таблицы>
[ALTER COLUMN <имя_столбца> [SET DEFAULT <выражение>]|
            [DROP DEFAULT]]
|[ADD <определение_столбца>]
|[DROP COLUMN <имя_столбца> [CASCADE]|[RESTRICT]]
|[ADD
[<определение_первичного_ключа>]|[<определение_внешнего_ключа>]|[<
условие_уникальности>]|[<условие_на_значение>]]
|[DROP CONSTRAINT <имя_ограничения> [CASCADE]|[RESTRICT]]
```

Команда ALTER TABLE берет на себя все действия по копированию данных во временную таблицу, удалению старой таблицы, созданию вместо нее новой таблицы с нужной структурой и последующим переписыванием в нее данных.

Назначение многих параметров и ключевых слов команды ALTER TABLE аналогично назначению соответствующих параметров и ключевых слов команды CREATE TABLE (например, синтаксис конструкции <определение_столбца> совпадает с синтаксисом аналогичной конструкции команды CREATE TABLE).

Основные режимы использования команды ALTER TABLE следующие:

- добавление столбца;
- удаление столбца;
- модификация столбца;
- изменение, добавление и удаление ограничений (первичных и внешних ключей, значений по умолчанию).

Добавление столбца

Для добавления нового столбца следует использовать ключевое слово ADD, после которого должно стоять определение столбца.

Добавим, например, в таблицу «Студенты» столбец *Год_поступления* следующим образом:

```
ALTER TABLE Студенты
ADD Год_поступления INTEGER NOT NULL DEFAULT
YEAR(GETDATE())
```

После выполнения этой команды в структуру таблицы «Студент» будет добавлен еще один столбец со значением по умолчанию, равным текущему году (значение по умолчанию вычисляется с помощью двух встроенных функций — YEAR() и GETDATE()).

Модификация столбца

Для модификации существующего столбца таблицы служит ключевое слово ALTER COLUMN. Изменение свойств столбца невозможно, если:

- столбец участвует в ограничениях PRIMARY KEY или FOREIGN KEY;
- на столбец наложены ограничения целостности CHECK или UNIQUE (исключение составляют столбцы, имеющие тип данных переменной длины, т. е. типы данных, начинающиеся на var);
- со столбцом связано значение по умолчанию (в этом случае допускается изменение длины, общего количества цифр или количества цифр после десятичной точки при неизменном типе данных).

Определяя для столбца новый тип данных, следует помнить о том, что старый тип данных должен конвертироваться в новый.

Пример модификации столбца «Номер_группы» таблицы «Студенты» (тип данных INTEGER заменяется на CHAR):

```
ALTER TABLE Студенты
ALTER COLUMN Номер_группы CHAR(6) NOT NULL
```

Удаление столбца

Для удаления столбца из таблицы используется предложение DROP COLUMN <имя_столбца>. При удалении столбцов следует учитывать, что нельзя удалять столбцы с ограничениями целостности CHECK, FOREIGN KEY, UNIQUE или PRIMARY KEY, а также столбцы, для которых определены значения по умолчанию (в виде ограничения целостности на уровне столбца или на уровне таблицы).

Рассмотрим, например, команду удаления из таблицы «Студенты» столбца «Год_поступления»:

```
ALTER TABLE Студенты
DROP COLUMN Год_поступления
```

Эта команда выполнена не будет, так как при добавлении столбца было определено значение по умолчанию.

Добавление ограничений на уровне таблицы

Для добавления ограничений на уровне таблицы используется предложение ADD CONSTRAINT <имя_ограничения>.

В качестве примера рассмотрим команды добавления внешних ключей в таблицы базы данных «Сессия».

Добавление внешних ключей в таблицу «Учебный_план» (создание связи с именем FK_Дисциплина и связи с именем FK_ Кадровый_состав):

```
ALTER TABLE Учебный_план
ADD CONSTRAINT FK_Дисциплина
FOREIGN KEY (ID_Дисциплина)
REFERENCES Дисциплины

ALTER TABLE Учебный_план
ADD CONSTRAINT FK_Кадровый_состав
FOREIGN KEY (ID_Преподаватель)
REFERENCES Кадровый_состав
```

Добавление внешних ключей в таблицу «Сводная_ведомость» (создание связи с именем FK_Студент и связи с именем FK_План):

```
ALTER TABLE Сводная_ведомость
ADD CONSTRAINT FK_Студент
FOREIGN KEY (ID_Студент)
REFERENCES Студенты

ALTER TABLE Сводная_ведомость
ADD CONSTRAINT FK_План
FOREIGN KEY (ID_План)
REFERENCES Учебный_план
```

С помощью конструкции ADD CONSTRAINT создается *поименованное* ограничение. Необходимо отметить, что удаление любого ограничения на уровне таблицы происходит только по его имени, поэтому ограничение должно быть поименовано (чтобы его можно было удалить).

Рассмотрим еще один пример — добавление значения по умолчанию для столбца *Номер_группы*:

```
ALTER TABLE Студент
ADD CONSTRAINT DEF_Номер_группы DEFAULT 1 FOR
Номер_группы
```

В результате выполнения этой команды на уровне таблицы будет создано ограничение целостности с именем DEF_Номер_группы.

Удаление ограничений

Для удаления из таблицы ограничения целостности используется предложение DROP CONSTRAINT <имя_ограничения>.

Удаление ограничения целостности возможно только в том случае, когда оно поименовано (т. е. предложение <определение_ограничения> содержит именование ограничения CONSTRAINT).

Команда удаления построенного внешнего ключа FK_Дисциплина из таблицы «Учебный_план» выглядит следующим образом:

```
ALTER TABLE Учебный_план
DROP CONSTRAINT FK_Дисциплина
```

Удалить же построенное ограничение DEF_Номер_группы можно с помощью следующей команды:

```
ALTER TABLE Студент
DROP CONSTRAINT DEF_Номер_группы
```

9.3.3. Удаление таблиц — команда DROP TABLE

Удаление таблицы выполняется при помощи команды DROP TABLE:

```
DROP TABLE <имя_таблицы>
```

Единственный аргумент команды задает имя таблицы, которую необходимо удалить.

Операция удаления таблицы в некоторых случаях требует определенного внимания. Невозможно удалить таблицу, если на нее с помощью ограничения целостности FOREIGN KEY ссылается другая таблица: попытка удаления таблицы «Дисциплины» вызовет сообщение об ошибке, так как на таблицу «Дисциплины» ссылается таблица «Учебный_план». Например, в ответ на использование команды:

```
DROP TABLE Дисциплины
```

будет выдано сообщение об ошибке, гласящее, что невозможно удалить таблицу, поскольку есть ограничение целостности FOREIGN KEY, ссылающееся на таблицу «Дисциплины».

9.4. Управление данными

Целью любой системы управления базами данных в конечном счете является ввод, изменение, удаление и выборка данных. Рассмотрим методы управления данными с помощью языка SQL.

9.4.1. Извлечение данных — команда SELECT

Основным инструментом выборки данных в языке SQL является команда SELECT. С помощью этой команды можно получить доступ к данным, представленным как совокупность таблиц практически любой сложности.

Чаще всего используется упрощенный вариант команды SELECT, имеющий следующий синтаксис:

```
SELECT <Список_выбора>
[ INTO <Новая_таблица> ]
FROM <Исходная_таблица>
[ WHERE <Условие_отбора> ]
[ GROUP BY <Ключи_группировки> ]
[ HAVING <Условие_отбора> ]
[ ORDER BY <Ключи_сортировки> [ ASC | DESC ] ]
```

Инструкция SELECT разбивается на отдельные разделы, каждый из которых имеет свое назначение. Из приведенного синтаксического описания видно, что обязательными являются только разделы SELECT и FROM, а остальные разделы могут быть опущены. Полный список разделов следующий:

```
SELECT    UNION
INTO      ORDER BY
FROM      COMPUTE
WHERE     FOR
GROUP BYOPTION
HAVING
```

Раздел SELECT

Основное назначение раздела SELECT (одного из двух обязательных разделов, которые должны указываться в любом запросе) — задание набора столбцов, возвращаемых после выполнения запроса, т. е. внешнего вида результата. В простейшем случае возвращается стол-

бец одной из таблиц, участвующих в запросе. В более сложных ситуациях набор значений в столбце формируется как результат вычисления выражения. Такие столбцы называются *вычисляемыми* и по умолчанию им не присваивается никакого имени.

При необходимости пользователь может указать для столбца, возвращаемого после выполнения запроса, произвольное имя. Такое имя называется *псевдонимом* (alias). В обычной ситуации назначение псевдонима необязательно, но в некоторых случаях требуется явное его указание. Наиболее часто это требуется при работе с разделом INTO, в котором каждый из возвращаемых столбцов должен иметь имя, и это имя должно быть уникально.

Помимо сказанного, с помощью раздела SELECT можно ограничить количество строк, которое будет включено в результат выборки. Синтаксис раздела SELECT следующий:

```
SELECT [ ALL | DISTINCT ]
[ TOP n [ PERCENT ] [ WITH TIES ] ]
<Список_выбора>
```

Рассмотрим назначение параметров.

Ключевые слова ALL | DISTINCT

При указании ключевого слова ALL в результат запроса выводятся *все* строки, удовлетворяющие сформулированным условиям, тем самым разрешается включение в результат одинаковых строк (одинаковость строк определяется на уровне результата отбора, а не на уровне исходных данных). Параметр ALL используется по умолчанию.

Если в запросе SELECT указывается ключевое слово DISTINCT, то в результат выборки не будет включаться более одной повторяющейся строки. Таким образом, каждая возвращенная строка будет уникальной. Уникальность строки при этом определяется на уровне строк результата выборки, а не на уровне исходных данных. Если в результат выборки включаются два столбца, уникальность будет определяться по значениям обоих этих столбцов. В отдельности значения в первом и втором столбцах могут повторяться, но комбинация значений в обоих столбцах должна быть уникальна. Аналогичные правила действуют и в отношении большего количества столбцов.

Рассмотрим результат использования ключевых слов ALL и DISTINCT на примере выборки столбцов *Семестр* и *Отчетность* из

Семестр	Отчетность
1	з
2	з
3	з
4	з
8	з
1	з
4	з
1	з
2	з
3	з
2	з
3	з
2	з
1	з
2	з
1	з
1	з
1	з
1	з

Семестр	Отчетность
1	з
1	з
2	з
2	з
3	з
3	з
4	з
4	з
5	з
5	з
6	з
6	з
7	з
7	з
8	з
8	з
9	з
9	з

Рис. 9.4. Действие ключевых слов:
а — ALL; *б* — DISTINCT

таблицы «Учебный_план» базы данных «Сессия» (рис. 9.4). Сначала выполним запрос с указанием ключевого слова ALL:

```
SELECT ALL Семестр, Отчетность
FROM Учебный_план
```

Фрагмент результата представлен на рис. 9.4, *а*.
Теперь заменим ключевое слово ALL на DISTINCT:

```
SELECT DISTINCT ALL Семестр, Отчетность
FROM Учебный_план
```

В этом случае результат запроса, представленный на рис. 9.4, *б* — это строки, содержащие одинаковые значения в столбцах, включенные только один раз. Этот результат должен свидетельствовать только о наличии различных форм отчетности в семестрах.

Ключевое слово TOP n [PERCENT] [WITH TIES]

Использование ключевого слова TOP n, где n — числовое значение, позволяет отобрать в результат не все строки, а только n первых. При этом выбираются первые строки результата выборки, а не исходных данных. Поэтому набор строк в результате выборки при указании ключевого слова TOP может меняться в зависимости от порядка сор-

тировки. Если в запросе используется раздел WHERE, то ключевое слово TOP работает с набором строк, возвращенных после применения логического условия, определенного в разделе WHERE.

Продемонстрируем использование ключевого слова TOP:

```
SELECT TOP 5 * FROM Студенты
```

В этом примере из таблицы «Студенты» базы данных «Сессия» было выбрано 5 первых строк (рис. 9.5).

ID_Студент	Фамилия	Имя	Отчество	Номер_Группы	Год_поступления
1	Агапов	Иван	Иванович	2002\1	2002
2	Агупов	Петр	Александрович	2000\1	2000
3	Агуреев	Дмитрий	Александрович	2002\1	2002
4	Акатьева	Мария	Алексеевна	2000\1	1999
5	Акулов	Алексей	Юрьевич	2000\2	2000

Рис. 9.5. Из таблицы «Студенты» выбрано пять строк

Можно также выбирать не фиксированное количество строк, а определенный процент от всех строк, удовлетворяющих условию. Для этого необходимо добавить ключевое слово PERCENT:

```
SELECT TOP 10 PERCENT * FROM Студенты
```

Всего в таблице было 115 строк, следовательно, 10 % будет составлять 11,5 строки. В результате будут выданы 12 строк (рис. 9.6).

ID_Студент	Фамилия	Имя	Отчество	Номер_Группы	Год_поступления
1	Агапов	Иван	Иванович	2002\1	2002
2	Агупов	Петр	Александрович	2000\1	2000
3	Агуреев	Дмитрий	Александрович	2002\1	2002
4	Акатьева	Мария	Алексеевна	2000\1	1999
5	Акулов	Алексей	Юрьевич	2000\2	2000
6	Алексеев	Иван	Александрович	2002\2	2002
7	Амаев	Тамерлан	Джабраилович	2001\2	2000
8	Аюбова	Оксана	Аюбовна	2001\2	2001
9	Барыкин	Юрий	Владимирович	2000\2	2000
10	Басов	Константин	Викторович	2000\2	2000
11	Бебчук	Мария	Борисовна	2000\1	2000
12	Белова	Ирина	Владимировна	2002\2	2002

Рис. 9.6. Из таблицы «Студенты» выбрано 10 % строк

Если указанное количество процентов строк представляет собой нецелое число, то сервер всегда выполняет округление в большую сторону.

Приведем также пример, демонстрирующий влияние порядка сортировки на возвращаемый набор строк:

```
SELECT TOP 10 PERCENT * FROM Студенты ORDER BY
Номер_Группы
```

В результате выполнения такого запроса будут выданы следующие 12 строк (рис. 9.7).

При указании вместе с предложением ORDER BY ключевого слова WITH TIES в результат будут включены еще и строки, совпадающие по значению колонки сортировки с последними выведенными строками запроса SELECT TOP n [PERCENT].

ID_Студент	Фамилия	Имя	Отчество	Номер_Группы	Год_поступления
2	Агупов	Петр	Александрович	2000\1	2000
4	Акатьева	Мария	Алексеевна	2000\1	1999
11	Бебчук	Мария	Борисовна	2000\1	2000
26	Голдобин	Михаил	Александрович	2000\1	2000
30	Гулько	Александр	Юрьевич	2000\1	2000
31	Гунько	Екатерина	Сергеевна	2000\1	2000
55	Кривченков	Михаил	Юрьевич	2000\1	2000
58	Лазаренко	Екатерина	Владимировна	2000\1	2000
61	Лебедев	Андрей	Евгеньевич	2000\1	2000
64	Лейпунская	Анна	Михайловна	2000\1	2000
73	Маслова	Анна	Владимировна	2000\1	2000
76	Митина	Светлана	Олеговна	2000\1	2000

Рис. 9.7. Выбор строк по номеру группы

ID_Студент	Фамилия	Имя	Отчество	Номер_Группы	Год_поступления
2	Агупов	Петр	Александрович	2000\1	2000
4	Акатьева	Мария	Алексеевна	2000\1	1999
11	Бебчук	Мария	Борисовна	2000\1	2000
26	Голдобин	Михаил	Александрович	2000\1	2000
30	Гулько	Александр	Юрьевич	2000\1	2000
31	Гунько	Екатерина	Сергеевна	2000\1	2000
55	Кривченков	Михаил	Юрьевич	2000\1	2000
58	Лазаренко	Екатерина	Владимировна	2000\1	2000
61	Лебедев	Андрей	Евгеньевич	2000\1	2000
64	Лейпунская	Анна	Михайловна	2000\1	2000
73	Маслова	Анна	Владимировна	2000\1	2000
76	Митина	Светлана	Олеговна	2000\1	2000
79	Никольская	Анастасия	Александровна	2000\1	2000
81	Панков	Дмитрий	Геннадьевич	2000\1	2000
83	Пасенова	Медея	Гераклитовна	2000\1	2000
84	Петрова	Алина	Николаевна	2000\1	2000
92	Сибгатуллина	Таисия	Насимовна	2000\1	2000
95	Соловьев	Антон	Алексеевич	2000\1	2000
99	Стыценко	Илья	Владимирович	2000\1	2000
104	Фролова	Елена	Владимировна	2000\1	2000
111	Шваб	Кирилл	Юрьевич	2000\1	2000

Рис. 9.8. Информация о всех студентах первой группы

Использование ключевого слова WITH TIES в предыдущем примере позволит обеспечить выдачу в ответ на запрос информации обо *всех* студентах первой по порядку группы:

```
SELECT TOP 10 PERCENT WITH TIES *
FROM Студенты
ORDER BY Номер_группы
```

После выполнения запроса получаем следующий результат (рис. 9.8).

Предложение <Список_выбора>

Синтаксис предложения <Список_выбора> следующий:

```
<Список_выбора> ::=
{ *
| { <Имя_таблицы> | <Псевдоним_таблицы> }.*
| { <Имя_столбца> | <Выражение> }
[ [ AS ] <Псевдоним_столбца>]
| <Псевдоним_столбца> = <Выражение>
} [ ,...,n ]
```

Символ «*» означает включение в результат всех столбцов, имеющихся в списке таблиц раздела FROM.

Если в результат не нужно включать все столбцы *всех* таблиц, то можно явно указать имя объекта, из которого необходимо выбрать все столбцы (<Имя_таблицы>.* или <Псевдоним_таблицы>.*).

Отдельный столбец таблицы в результат выборки включается явным указанием имени столбца (параметр <Имя_столбца>). Столбец должен принадлежать одной из таблиц, указанных в разделе FROM. Если столбец с указанным именем имеется более чем в одном источнике данных, перечисленных в разделе FROM, то необходимо явно указать имя источника данных, к которому принадлежит столбец в формате <Имя_таблицы>.<Имя_столбца>. В противном случае будет выдано сообщение об ошибке.

Например, попробуем выбрать данные из столбца *ID_Дисциплина*, который имеется в таблицах «Дисциплина» и «Учебный_план»:

```
SELECT ID_Дисциплина, Наименование, Семестр
FROM Дисциплина, Учебный_план
```

В ответ будет выдано сообщение об ошибке, указывающее на некорректное использование имени *ID_Дисциплина*.

То есть в этом случае необходимо явно указать имя источника данных, которому принадлежит столбец, например:

```
SELECT Дисциплина.ID_Дисциплина, Наименование, Семестр
FROM Дисциплина, Учебный_план
```

Столбцам, возвращаемым как результат выполнения запроса, могут быть присвоены *псевдонимы*. Псевдонимы позволяют изменить имя исходного столбца или поименовать столбец, содержимое которого получено как результат вычисления выражения. Имя псевдонима указывается с помощью параметра [AS] <Псевдоним_столбца>. Ключевое слово AS необязательно при задании псевдонима. В общем случае сервер не требует уникальности имен столбцов результата выборки, поэтому разные столбцы могут иметь одинаковые имена или псевдонимы.

Столбцы в результате выборки могут быть не только копией столбца одной из исходных таблиц, но и формироваться на основе вычисления выражения. Такой столбец в списке выбора задается с помощью конструкции <Выражение> [[AS] <Псевдоним_столбца>]. Выражение при этом может содержать константы, имена столбцов, функции, а также их комбинации. Дополнительно столбцу, формируемому на основе вычисления выражения, можно присвоить псевдоним, указав его с помощью параметра [AS] <Псевдоним_столбца>. По умолчанию вычисляемый столбец не имеет имени.

Другой способ формирования вычисляемого столбца состоит в использовании конструкции со знаком равенства: <Псевдоним_столбца> = <Выражение>. Единственным отличием этого способа от предыдущего является необходимость *обязательного* задания псевдонима. В простейшем случае выражение является именем столбца, константой, переменной или функцией. Если в качестве выражения выступает имя столбца, то получаем еще один способ задания псевдонима для столбца.

Рассмотрим следующий пример. Пусть для таблицы «Студенты» необходимо построить запрос, представляющий фамилию, имя и отчество в одной колонке. Используя операцию конкатенации (сложения) символьных строк и значение ФИО в качестве псевдонима столбца, построим запрос:

```
SELECT TOP 10 Фамилия + ' ' + Имя + ' ' + Отчество as ФИО,
              Номер_Группы
FROM Студенты
```

Результат запроса показан на рис. 9.9.

ФИО	Группа
Агапов Иван Иванович	2002\1
Агупов Петр Александрович	2000\1
Агуреев Дмитрий Александрович	2002\1
Акатьева Мария Алексеевна	2000\1
Акулов Алексей Юрьевич	2000\2
Алексеев Иван Александрович	2002\2
Амаев Тамерлан Джабраилович	2001\2
Аюбова Оксана Аюбовна	2001\2
Барыкин Юрий Владимирсвич	2000\2
Басов Константин Викторович	2000\2

Рис. 9.9. Результат запроса — фамилия, имя и отчество в одной колонке

Раздел FROM

С помощью раздела FROM определяются источники данных, с которыми будет работать запрос.

Синтаксис раздела FROM следующий:

```
FROM { <Источник_данных> } [,...,n]
```

На первый взгляд конструкция раздела выглядит простой. Однако при ближайшем рассмотрении он оказывается довольно сложным. В основном работа с разделом FROM — это перечисление через запятую источников данных, с которыми должен работать запрос. Собственно источник данных указывается с помощью предложения <Источник_данных>, синтаксис которого следующий:

```
<Источник_данных> ::= <имя_таблицы> [ [AS] <псевдоним_таблцы>]|
<связка_таблиц>
```

С помощью параметра <имя_таблицы> указывается имя обычной таблицы. Параметр <псевдоним_таблицы> используется для присвоения таблице псевдонима, под которым на нее нужно будет ссылаться в запросе. Часто псевдонимы таблиц применяют, чтобы ссылку на нужную таблицу сделать более удобной и короткой. Например, если в запросе часто упоминается имя таблицы «Учебный_план», то можно воспользоваться псевдонимом, например, tpl. Указание ключевого слова AS не является при этом обязательным.

Конструкция <связка_таблиц> реализует один из наиболее сложных методов задания источника данных. С помощью нее можно связать данные двух и более таблиц в единый набор данных, указав критерии связывания. Синтаксис конструкции <связка_таблиц> следующий:

```
<связка_таблиц> ::= <левая_таблица> <тип_связывания> <правая_таблица>
        ON <условие_связывания>
```

Конструкция <тип_связывания> описывает тип связывания двух таблиц. Исходная таблица указывается слева от конструкции <тип_связывания> (<левая_таблица>), а справа указывается зависимая таблица (<правая_таблица>).

Общий синтаксис конструкции <тип_связывания> следующий:

```
<тип_связывания> ::= [INNER | {{LEFT | RIGHT | FULL }
    [OUTER] } ] JOIN
```

Как видно, обязательным в конструкции является ключевое слово JOIN.

Конструкция ON <условие_связывания> задает логическое условие связывания двух таблиц. Допустимы операторы сравнения (например, =, <, >, <=, >=, !-, <>). Чаще всего используется оператор равенства, например:

```
ON Учебный_план.ID_Дисциплина =
Дисциплины.ID_Дисциплина
```

В этом примере устанавливается связь между таблицами «Учебный_план» и «Дисциплина» по столбцу *ID_Дисциплина*, имеющемуся в каждой из таблиц.

Ключевое слово INNER

Этот тип связи используется по умолчанию. Указание сочетания INNER JOIN равносильно указанию только ключевого слова JOIN. В качестве кандидатов на включение в результат запроса рассматриваются пары строк, удовлетворяющие критерию связывания в обеих таблицах. Затем строки из левой таблицы, для которых не имеется пары в связанной таблице, в результат не включаются. Также не включаются в результат и строки правой таблицы, для которых нет соответствующей строки в левой таблице.

В приведенном ниже примере выполняется выборка данных из таблиц «Дисциплины» и «Учебный_план» с помощью запроса SELECT. Таблицы связаны по ключевому полю *ID_Дисциплина*, имеющемуся в каждой из них. Для каждой строки таблицы «Учебный_план» ищется строка с совпадающим значением поля *ID_Дисциплина* в таблице «Дисциплины». Все строки таблицы «Учебный_план», для которых нет строк с соответствующим значением поля *ID_Дисциплина*, игнорируются и не включаются в конечный результат. Аналогично не включаются в результат все строки таблицы «Дисциплины», для которых нет соответствующей строки в таблице «Учебный_план» (что, однако, невозможно для данного примера, так как столбец *ID_Дисциплина* таблицы «Учебный_план» связан внешним ключом со столбцом *ID_Дисциплина* таблицы «Дисциплины»).

```
SELECT Наименование, Семестр, Количество_часов
FROM Учебный_план INNER JOIN Дисциплины ON
    Учебный_план.ID_Дисциплина =
Дисциплины.ID_Дисциплина
WHERE Количество_часов > 60
```

В результате выполнения этой команды будет возвращен набор строк, изображенный на рис. 9.10.

	Наименование	Семестр	Количество_часов
▶	Английский язык	1	90
	Английский язык	2	110
	Английский язык	3	90
	Английский язык	4	100
	Английский язык	8	80
	История России	1	72
	Физическая культура	1	138
	Физическая культура	2	130
	Физическая культура	3	140
	Информатика	1	90
*			

Рис. 9.10. Дисциплины, входящие в учебный план

Ключевое слово LEFT [OUTER]

При использовании ключевого слова LEFT в результат будут включены все строки левой таблицы, независимо от того, есть для них соответствующая строка в правой таблице или нет. В случае отсутствия строки в правой таблице для столбцов правой таблицы, включенных в результат выборки, устанавливается значение NULL.

В приведенном ниже примере иллюстрируется использование ключевого слова LEFT [OUTER] для выборки данных.

```
SELECT Наименование, Семестр, Отчетность
FROM Дисциплины LEFT OUTER JOIN Учебный_план ON
     Учебный_план.ID_Дисциплина =
Дисциплины.ID_Дисциплина
WHERE (Наименование LIKE '%информатик%')
```

Будет возвращен набор строк, изображенный на рис. 9.11.

Наименование	Семестр	Отчетность
Информатика и программирование	<NULL>	<NULL>
Информатика	1	з
Высокоуровневые методы информатики и программиров	4	з
Высокоуровневые методы информатики и программиров	4	з
*		

Рис. 9.11. Все дисциплины, названия которых содержат подстроку 'Информатик'

Как видно, по сравнению с использованием ключевого слова INNER, в результат запроса добавлена строка из таблицы «Дисциплины», которая удовлетворяет сформулированному условию отбора, но для которой не существует соответствующей строки в таблице «Учебный_план». В столбцах *Семестр* и *Отчетность* (относящихся к таблице «Учебный_план») для этих строк установлено значение NULL.

Ключевое слово RIGHT [OUTER]

При использовании этого ключевого слова в результат будут включены все строки правой таблицы независимо от того, есть ли для них соответствующая строка в левой таблице. Для соответствующих столбцов левой таблицы, включенных в запрос, устанавливается значение NULL. Приведем пример такого запроса:

```
SELECT Отчетность, Семестр, Наименование
FROM Учебный_план RIGHT OUTER JOIN Дисциплины ON
     Учебный_план.ID_Дисциплина =
Дисциплины.ID_Дисциплина
WHERE (Наименование LIKE '%информатик%')
```

Этот пример основывается на тех же данных, что и предыдущий, но связь таблиц устанавливается в обратном порядке. После выпол-

	Отчетность	Семестр	Наименование
	<NULL>	<NULL>	Информатика и программирование
	э	1	Информатика
	з	4	Высокоуровневые методы информатики и программиров
▶	э	4	Высокоуровневые методы информатики и программиров
*			

Рис. 9.12. Связь таблиц «Дисциплина» и «Учебный план»

нения приведенной инструкции будет получен результат, показанный на рис. 9.12.

Ключевое слово FULL [OUTER]

При использовании ключевого слова FULL в результат будут включены все строки как правой, так и левой таблицы. Применение ключевого слова FULL [OUTER] можно рассматривать как одновременное применение ключевых слов LEFT [OUTER] и RIGHT [OUTER].

Раздел WHERE

Раздел WHERE предназначен для наложения вертикальных фильтров на данные, обрабатываемые запросом. Другими словами, с помощью раздела WHERE можно сузить набор строк, включаемых в результат выборки. Для этого указывается логическое условие, от которого зависит, будет ли строка включена в выборку по запросу или нет. Строка включается в результат выборки, только если логическое выражение возвращает значение TRUE.

В общем случае логическое выражение содержит имена столбцов таблиц, с которыми работает запрос. Для каждой строки, возвращенной запросом, вычисляется логическое выражение путем подстановки вместо имен столбцов конкретных значений из соответствующей строки. Если при вычислении выражения возвращается значение TRUE, то есть выражение истинно, то строка будет включена в конечный результат. В противном случае строка в результат не включается. При необходимости можно указать более одного логического выражения, объединив их с помощью логических операторов OR и AND.

Рассмотрим синтаксис раздела WHERE.

```
WHERE <условие_отбора>
| <имя_столбца> {= | *= | =*} <имя_столбца>
```

В конструкции <условие_отбора> можно определить любое логическое условие, при выполнении которого строка будет включена в

результат. Хотя и было сказано, что обычно логическое условие содержит имена столбцов, оно может быть и произвольным, в том числе и совсем не связанным с данными. Например, в следующей команде условие WHERE никогда не выполнится и ни одна строка не будет возвращена:

```
SELECT * FROM Дисциплины WHERE 3=5
```

Приведенный пример демонстрирует логику работы раздела WHERE. Более удачное использование логического условия приведено в следующем примере:

```
SELECT Фамилия, Имя, Отчество, Номер_Группы,
       Год_поступления
FROM Студенты
WHERE Год_поступления < 2000
```

В результате будет возвращен список всех студентов, поступивших на факультет ранее 2000 г. (рис. 9.13).

Фамилия	Имя	Отчество	Номер_Группы	Год_поступления
Акатьева	Мария	Алексеевна	2000\1	1999
Козлова	Нина	Сергеевна	2000\2	1999

Рис. 7.13. Студенты, поступившие на факультет ранее 2000 г.

Помимо операций сравнения (=, >, <, >=, <=) и логических операторов OR, AND, NOT при формировании условия отбора могут быть использованы дополнительные логические операторы, расширяющие возможности управления данными. Рассмотрим некоторые из этих операторов.

Оператор BETWEEN

С помощью этого оператора можно определить, лежит ли значение указанной величины в заданном диапазоне. Синтаксис использования оператора следующий:

```
<выражение> [NOT] BETWEEN <начало_диапазона>
      AND <конец_диапазона>
```

<Выражение> задает проверяемую величину, а аргументы <начало_диапазона> и <конец_диапазона> определяют возможные границы ее изменения. Использование оператора NOT совместно с опера-

тором BETWEEN позволяет задать диапазон, *вне* которого может изменяться проверяемая величина.

При выполнении оператор BETWEEN преобразуется в конструкцию из двух операций сравнения:

```
(<выражение> >= <начало_диапазона>)
    AND (<выражение> <= <конец_диапазона>)
```

Рассмотрим пример использования оператора BETWEEN:

```
SELECT Наименование, Семестр, Количество_часов
FROM Учебный_план INNER JOIN
     Дисциплины ON
     Учебный_план.ID_Дисциплина =
Дисциплины.ID_Дисциплина
WHERE Количество_часов BETWEEN 50 AND 100
```

В результате выполнения инструкции получим список дисциплин учебного плана с количеством часов от 50 до 100 (рис. 9.14).

Наименование	Семестр	Количество_часов
Английский язык	1	90
Английский язык	3	90
Английский язык	4	100
Английский язык	8	80
История России	1	74
Концепции современного естествознания	3	56
Математический анализ	2	56
Дискретная математика	4	54
Основы алгебры и геометрии	1	52
Основы алгебры и геометрии	2	52
Информатика	1	92
Основы программирования	1	52
Теория вероятности и математическая статистика	3	54
Операционная система UNIX	7	56
Финансы и кредит	5	50
Менеджмент	5	50
Статистика	8	50
Предметно-ориентированные информационные системы	8	50

Рис. 9.14. Список дисциплин учебного плана с количеством часов от 50 до 100

Оператор IN

Оператор позволяет задать в условии отбора множество возможных значений для проверяемой величины. Синтаксис использования оператора следующий:

```
<выражение> [NOT] IN (<выражение1>,...,<выражениеN>)
```

<Выражение> указывает проверяемую величину, а аргументы <выражение1>,..., <выражениеN> задают перечислением через запятую набор значений, которые может принимать проверяемая величина. Ключевое слово NOT выполняет логическое отрицание.

Рассмотрим пример применения оператора IN.

```
SELECT Наименование, Семестр, Количество_часов
FROM Учебный_план INNER JOIN
     Дисциплины ON
     Учебный_план.ID_Дисциплина =
Дисциплины.ID_Дисциплина
WHERE Наименование IN ('Английский язык', 'Физическая
      культура')
```

В результате выполнения инструкции получим строки учебного плана для дисциплин «Английский язык» и «Физическая культура» (рис. 9.15).

Наименование	Семестр	Количество_часов
Английский язык	1	90
Английский язык	2	110
Английский язык	3	90
Английский язык	4	100
Английский язык	8	80
Физическая культура	1	138
Физическая культура	2	130
Физическая культура	3	140

Рис. 9.15. Выбраны строки для дисциплин «Английский язык» и «Физкультура»

Оператор LIKE

С помощью оператора LIKE можно выполнять сравнение выражения символьного типа с заданным шаблоном. Синтаксис оператора следующий:

```
<Символьное_выражение> [NOT] LIKE <образец>
```

<Образец> задает символьный шаблон для сравнения и заключается в кавычки. Шаблон может содержать символы-разделители. Допускается использование следующих символов-разделителей (табл. 9.10).

Рассмотрим пример использования оператора:

```
SELECT Фамилия, Имя, Отчество, Должность
FROM Кадровый_состав
WHERE Должность LIKE '%пр%'
```

Таблица 9.10

Символы-разделители	Значение
%	Может быть заменен в символьном выражении любым количеством произвольных символов. Например, образец '**%кош%**' позволяет отобрать слова: '**кошка**', '**окошко**', '**лукошко**', '**кошма**' и т. п.
_	Может быть заменен в символьном выражении любым, но только одним символом. Например, образец '**программ_**' позволяет отобрать слова: '**программа**', '**программ**', '**программы**', но не 'программист' или 'программой'
[ABC0-9]	Может быть заменен в символьном выражении только одним символом из указанного в квадратных скобках набора. Дефис используется для указания диапазона. Например, образец любой последовательности символов, начинающейся с буквы латинского алфавита, может быть задан следующим образом: '**[A-Z]%**'
[^ABC0-9]	Может быть заменен в символьном выражении только одним символом, кроме тех, что указаны в квадратных скобках. Дефис используется для указания диапазона. Например, образец любой последовательности символов, которая не должна заканчиваться цифрой, может быть задан следующим образом: '**%[^0-9]**'

Результат выполнения инструкции показан на рис. 9.16.

Фамилия	Имя	Отчество	Должность
Сидоров	Самуил	Савельевич	Проф.
Гиацинтова	Галина	Григорьевна	Проф.
Петров	Павел	Петрович	Ст. преп.
Китов	Кирилл	Кириллович	Проф.
Воробьева	Веса	Васильевна	Ст. преп.

Рис. 9.16. Выбраны профессора и преподаватели

Применение образца для значения столбца *Должность* в данном случае позволило отобрать строки со значениями «Ст. преп.» и «Проф».

Связывание таблиц

Раздел WHERE может быть использован для связывания таблиц. В этом случае условие связывания должно присоединяться к логическому выражению с помощью логической операции AND (логическое умножение).

Рассмотрим пример, уточняющий один из представленных выше:

```
SELECT Наименование, Семестр, Количество_часов
FROM Учебный_план INNER JOIN Дисциплины ON
     Учебный_план.ID_Дисциплина =
Дисциплины.ID_Дисциплина
WHERE (Количество_часов > 60) AND (Семестр = 1)
```

Перенесем условие связывания в логическое выражение:

```
SELECT Наименование, Семестр, Количество_часов
FROM Учебный_план, Дисциплины
WHERE (Учебный_план.ID_Дисциплина =
      Дисциплины.ID_Дисциплина) AND
      (Количество_часов > 60) AND (Семестр = 1)
```

Результат выполнения обоих запросов одинаков (рис. 9.17).

Наименование	Семестр	Количество_часов
Английский язык	1	90
История России	1	72
Физическая культура	1	138
Информатика	1	90

Рис. 9.17. Дисциплины первого семестра с количеством часов больше 60

Использование только условия связывания в разделе WHERE аналогично связыванию ключевым словом INNER в разделе FROM. Например, результаты следующих запросов одинаковы (рис. 9.18):

```
SELECT TOP 10 Наименование, Семестр, Количество_часов
FROM Учебный_план, Дисциплины
WHERE (Учебный_план.ID_Дисциплина =
      Дисциплины.ID_Дисциплина)

SELECT TOP 10 Наименование, Семестр, Количество_часов
FROM Учебный_план INNER JOIN Дисциплины ON
     Учебный_план.ID_Дисциплина =
Дисциплины.ID_Дисциплина
```

Содержимое обеих таблиц можно посмотреть с помощью следующих запросов:

```
SELECT TOP 10 *
FROM Учебный_план
```

[illegible]	Семестр	Количество часов
Английский язык	1	90
Английский язык	2	110
Английский язык	3	90
Английский язык	4	100
Английский язык	8	80
История России	1	72
Правоведение	4	32
Физическая культура	1	138
Физическая культура	2	130
Физическая культура	3	140

Рис. 9.18. Два запроса дают одинаковый результат

Результат примера — на рис. 9.19.

```
SELECT TOP 10 *
FROM Дисциплины
```

Результат примера — на рис. 9.20.

ID План	ID_Дисциплина	Семестр	Отчетно	[illegible]ество часов	ID_преподавателя
1	1	1	з	90	<NULL>
2	1	2	з	110	<NULL>
3	1	3	з	90	<NULL>
4	1	4	з	100	<NULL>
5	1	8	з	80	<NULL>
6	2	1	з	72	<NULL>
7	3	4	з	32	<NULL>
8	4	1	з	138	<NULL>
9	4	2	з	130	<NULL>
10	4	3	з	140	<NULL>

Рис. 9.19. Первые 10 строк из таблицы «Учебный план»

ID_Дисциплина	Наименование
1	Английский язык
2	История России
3	Правоведение
4	Физическая культура
5	Философия
6	Русский язык и культура речи
7	Экономическая теория
8	История мировых цивилизаций
9	Культурология
10	Социология

Рис. 9.20. Первые 10 строк из таблицы «Дисциплины»

Аналогом использования ключевых слов LEFT OUTER JOIN является указание в разделе WHERE условия с помощью символов *=. Приведенные примеры возвращают одинаковый набор данных:

```
SELECT Наименование, Семестр, Отчетность
FROM Дисциплины LEFT OUTER JOIN Учебный_план ON
     Учебный_план.ID_Дисциплина =
Дисциплины.ID_Дисциплина
WHERE (Наименование LIKE '%информатик%')

SELECT Наименование, Семестр, Отчетность
FROM Дисциплины, Учебный_план
WHERE (Учебный_план.ID_Дисциплина *=
      Дисциплины.ID_Дисциплина)
AND (Наименование LIKE '%информатик%')
```

Аналогом использования ключевых слов RIGHT OUTER JOIN является указание условия с помощью символов =*. Приведенные примеры возвращают одинаковый набор данных:

```
SELECT Отчетность, Семестр, Наименование
FROM Учебный_план RIGHT OUTER JOIN Дисциплины ON
     Учебный_план.ID_Дисциплина =
Дисциплины.ID_Дисциплина
WHERE (Наименование LIKE '%информатик%')

SELECT Отчетность, Семестр, Наименование
FROM Учебный_план, Дисциплины
WHERE (Учебный_план.ID_Дисциплина =*
      Дисциплины.ID_Дисциплина)
AND (Наименование LIKE '%информатик%')
```

Следует отметить, что при использовании специальных ключевых слов INNER | {LEFT | RIGHT | FULL } [OUTER] данные представляются по-иному, чем при указании условия WHERE. Скорость выполнения запроса в первом случае оказывается выше, поскольку организуется *связывание* данных, тогда как при использовании конструкции WHERE происходит их *фильтрация*. При выполнении запросов на небольших наборах данных это не играет существенной роли, поэтому удобнее обращаться к конструкции WHERE из-за наглядности и простоты синтаксиса этого варианта, но при построении сложных за-

просов, выполняющих обработку тысяч строк, все же лучше использовать конструкцию связывания.

Раздел ORDER BY

Раздел ORDER BY предназначен для упорядочения набора данных, возвращаемого после выполнения запроса. Рассмотрим пример упорядочения данных таблицы «Дисциплины» по столбцу *Наименование* в алфавитном порядке:

```
SELECT TOP 10 *
FROM Дисциплины
ORDER BY Наименование
```

Результат сортировки представлен на рис. 9.21.

ID_Дисциплина	Наименование
1	Английский язык
28	Базы данных
46	Бухгалтерский учет
36	Введение в специальность
33	Высокоуровневые методы информатики и программир
32	Вычислительные системы, сети и телекоммуникаций
16	Дискретная математика
53	Имитационное моделирование
48	Интеллектуальные информационные системы
19	Информатика

Рис. 9.21. Первые десять дисциплин расположены в алфавитном порядке

Полный синтаксис раздела ORDER BY следующий:

```
ORDER BY {<условие_сортировки> [ ASC | DESC ] } [,...,n]
```

Параметр <условие_сортировки> требует задания выражения, в соответствии с которым будет осуществляться сортировка строк. В простейшем случае это выражение представляет собой имя столбца одного из источников данных запроса.

Следует отметить, что в выражении, в соответствии с которым осуществляется сортировка строк, могут использоваться и столбцы, не указанные в разделе SELECT, то есть не входящие в результат выборки.

Раздел ORDER BY разрешает использование ключевых слов ASC и DESC, с помощью которых можно явно указать, каким образом

следует упорядочить строки. При указании ключевого слова ASC данные будут отсортированы по возрастанию. Если необходимо отсортировать данные по убыванию, указывается ключевое слово DESC. По умолчанию используется сортировка по возрастанию.

Данные можно отсортировать по нескольким столбцам. Для этого необходимо ввести имена столбцов через запятую по порядку сортировки. Сначала данные сортируются по столбцу, имя которого было указано в разделе ORDER BY первым. Затем, если имеется множество строк с одинаковыми значениями в первом столбце, выполняется дополнительная сортировка этих строк по второму столбцу (внутри группы с одинаковым значением в первом столбце) и т. д.

Приведем пример сортировки по двум столбцам:

```
SELECT TOP 20 Наименование, Семестр, Количество_часов
FROM Учебный_план, Дисциплины
WHERE (Учебный_план.ID_Дисциплина =
      Дисциплины.ID_Дисциплина)
ORDER BY Семестр, Количество_часов DESC
```

Возвращаемый набор строк показан на рис. 9.22.

Наименование	Семестр	Количество_часов
Физическая культура	1	138
Английский язык	1	90
Информатика	1	90
История России	1	72
Основы алгебры и геометрии	1	52
Основы программирования	1	52
Культурология	1	32
Социология	1	32
Политология	1	32
Психология и педагогика	1	32
Математический анализ	1	26
Математический анализ	1	26
История мировых цивилизаций	1	20
Введение в специальность	1	12
Физическая культура	2	130
Английский язык	2	110
Математический анализ	2	56
Основы алгебры и геометрии	2	52
Экономическая теория	2	34
Философия	2	32

Рис. 9.22. Дисциплины сортируются по семестрам и затем по количеству часов в порядке убывания

Добавим в раздел SELECT столбец *Отчетность* и получим пример сортировки по трем столбцам:

```
SELECT TOP 20 Наименование, Семестр, Количество_часов,
              Отчетность
FROM Учебный_план, Дисциплины
WHERE (Учебный_план.ID_Дисциплина =
      Дисциплины.ID_Дисциплина)
ORDER BY Семестр, Отчетность, Количество_часов
```

Будет возвращен следующий набор строк, который показан на рис. 9.23.

Наименование	Семестр	Количество_часов	Отчетность
Введение в специальность	1	12	з
История мировых цивилизаций	1	20	з
Математический анализ	1	26	з
Культурология	1	32	з
Социология	1	32	з
Политология	1	32	з
Психология и педагогика	1	32	з
Основы алгебры и геометрии	1	52	з
Основы программирования	1	52	з
Английский язык	1	90	з
Физическая культура	1	138	з
Математический анализ	1	28	э
История России	1	74	э
Информатика	1	92	э
Основы программирования	2	26	з
Философия	2	32	з
Экономическая теория	2	34	з
Английский язык	2	110	з
Физическая культура	2	130	з
История мировых цивилизаций	2	20	з

Рис. 9.23. Сортировка по семестру, по количеству часов и форме отчетности

Раздел GROUP BY

Раздел GROUP BY позволяет выполнять группировку строк таблиц по определенным критериям. Для каждой группы можно выполнить специальные функции агрегирования, которые применяются ко всем строкам в группе. Одним из примеров использования раздела GROUP BY является суммирование однотипных значений.

Синтаксис раздела GROUP BY следующий:

```
GROUP BY [ALL] <условие_группировки> [,...,n]
```

При использовании группировки (раздела GROUP BY) на раздел SELECT накладываются дополнительные ограничения. В непосредственном виде разрешается указание только имен столбцов, перечисленных в разделе GROUP BY, то есть тех столбцов, по которым осуществляется группировка. Значения других столбцов не могут быть выведены в непосредственном виде, так как обычно каждая группа содержит множество строк, а в результате выборки для каждой группы должно быть указано единственное значение. Поэтому, чтобы вывести значения столбцов, не задающих критерии группировки, необходимо использовать функции агрегирования.

Аргумент <условие_группировки> определяет условие группировки. Обычно в качестве условия группировки указывается имя столбца, однако в общем случае разрешается использование и выражений, включающих ссылки на столбцы.

Для отбора в результат запроса групп, соответствующих некоторому условию, используется раздел HAVING. Синтаксис раздела совпадает с синтаксисом раздела WHERE.

Функции агрегирования позволяют выполнять статистическую обработку данных, подсчитывая количество, сумму, среднее значение и другие величины для всего набора данных. Во многих функциях агрегирования допускается использование ключевых слов ALL и DISTINCT. Ключевое слово ALL выполняет агрегирование всех строк исходного набора данных. При указании ключевого слова DISTINCT будет выполняться агрегирование только уникальных строк. Все повторяющиеся строки будут проигнорированы. По умолчанию выполняется агрегирование всех строк, то есть используется ключевое слово ALL. Далее приведены описания некоторых функций агрегирования.

AVG()

Эта функция вычисляет *среднее значение* для указанного столбца. Функция имеет следующий синтаксис:

```
AVG([ALL | DISTINCT] <выражение>)
```

При выполнении группировки (GROUP BY) вычисляется среднее значение для каждой группы. Если группировка не используется, то вычисляется среднее по всему столбцу. Например:

```
SELECT AVG(Количество_часов) FROM Учебный_план
```

Результат запроса:

```
-----------
41

(1 row(s) affected)
```

Теперь рассмотрим пример использования функции AVG совместно с разделом GROUP BY при выполнении группировки по столбцу *Семестр*:

```
SELECT Семестр, AVG(Количество_часов)
FROM Учебный_план
GROUP BY Семестр
```

Результат:

```
Семестр
----------- -----------
1           50
2           54
3           46
4           39
5           37
6           27
7           34
8           44
9           32

(9 row(s) affected)
```

COUNT()

Функция подсчитывает количество строк в группе (при выполнении группировки) или количество строк результата запроса. Синтаксис функции COUNT следующий:

```
COUNT({[ALL | DISTINCT] <выражение>] | *})
```

Параметр <выражение> в простейшем случае представляет собой имя столбца. Если обрабатываемая строка в соответствующем столбце содержит значение не NULL, то счетчик будет увеличен на единицу. Указание символа (*) предписывает считать *общее количество строк* независимо от того, содержат они значения NULL или нет.

Пример использования функции COUNT:

```
SELECT COUNT(*) AS 'Всего сотрудников',
COUNT(Телефон) AS 'С домашним телефоном'
FROM Кадровый_состав
```

Этот запрос подсчитывает общее количество строк в таблице, а также количество ненулевых значений в столбце *Телефон*.

Результат выполнения запроса:

```
Всего сотрудников С домашним телефоном
----------------- --------------------
14                10

(1 row(s) affected)

Warning: Null value eliminated from aggregate.
```

Пример использования функции COUNT() при выполнении группировки:

```
SELECT Должность, COUNT(*)
FROM Кадровый_состав
GROUP BY Должность
```

Данный запрос возвращает количество строк в каждой группе столбца *Должность*:

```
Должность
---------------------- ----------
Ассистент              3
Доцент                 4
Зав.каф.               2
Проф.                  3
Ст.преп.               2

(5 row(s) affected)
```

MAX()

Функция возвращает *максимальное* значение в указанном диапазоне. Эта функция может использоваться как в обычных запросах, так и в запросах с группировкой. Синтаксис функции следующий:

```
MAX([ALL | DISTINCT] <выражение>)
```

Пример использования функции:

```
SELECT MAX(Количество_часов),
       MAX(Количество_часов/2)
FROM Учебный_план
```

Результат выполнения запроса:

```
---------- ----------
140        70

(1 row(s) affected)
```

MIN()

Функция возвращает минимальное значение в указанном диапазоне. Синтаксис функции следующий:

```
MIN([ALL | DISTINCT] <выражение>)
```

Пример использования функции:

```
SELECT MIN(Количество_часов)
FROM Учебный_план
```

Результат выполнения запроса:

```
----------
12

      (1 row(s) affected)
```

SUM()

Функция выполняет обычное суммирование значений в указанном диапазоне. В качестве такого диапазона может рассматриваться группа или весь набор строк (без использования раздела GROUP BY).

Синтаксис функции следующий:

```
SUM([ALL | DISTINCT] <выражение>)
```

В качестве примера просто суммируем значения в столбце *Количество_часов*:

```
SELECT SUM(Количество_часов), COUNT(*),
     SUM(Количество_часов)/COUNT(*),
AVG(Количество_часов)
FROM Учебный_план
```

Результат выполнения запроса:

```
---------- ---------- ---------- ----------
694        89         41         41

(1 row(s) affected)
```

Теперь вновь обратимся к разделу SELECT и приведем пример группировки значений таблицы «Учебный_план». Произведем группировку строк по семестрам (столбец *Семестр*) и подсчитаем общую нагрузку в часах за каждый семестр:

```
SELECT Семестр, SUM(Количество_часов) AS 'Нагрузка'
FROM [Учебный_план]
GROUP BY Семестр
```

В результате выполнения запроса получен результат, показанный на рис. 9.24.

	Семестр	Нагрузка
▶	1	706
	2	486
	3	562
	4	474
	5	484
	6	334
	7	312
	8	266
	9	64
*		

Рис. 9.24. Группировка строк по семестрам с подсчетом общей нагрузки за каждый семестр

В первом столбце выведен номер семестра. Это единственный столбец исходной таблицы, который можно включать в запрос непосредственно, так как по нему осуществляется группировка. Во втором столбце с помощью функции агрегирования SUM была получена сумма значений столбца *Количество_часов*. Функции агрегирования работают со всеми строками группы, возвращая единственное значение для всех этих строк.

Рассмотрим теперь запрос, подсчитывающий количество экзаменов в каждом семестре:

```
SELECT Семестр, COUNT(*) AS 'Экзамены'
FROM [Учебный_план]
WHERE Отчетность = 'э'
GROUP BY Семестр
```

Результат выполнения запроса показан на рис. 9.25.

Семестр	Экзамены
1	3
2	4
3	5
4	6
5	4
6	5
7	3
8	3
9	1

Рис. 9.25. Количество экзаменов в каждом семестре

Предложение группировки может содержать ключевое слово ALL. Назначение этого слова следующее. Нередко при выполнении группировки используется раздел WHERE, то есть группировка должна выполняться не над всеми строками, а лишь над определенной частью строк. Результатом такого подхода может явиться то, что одна или более групп не будет содержать ни одной строки. Если группа не содержит ни одной строки, то по умолчанию эта группа не включается в результат выборки. Однако в некоторых ситуациях все же требуется, чтобы были выведены все группы, в том числе и не содержащие ни одной строки. Для этого и необходимо указывать в разделе GROUP BY ключевое слово ALL. В этом случае будет выводиться список всех групп, но для групп, не содержащих строк, не будут выполняться функции агрегирования.

Рассмотрим это на примере. Для начала выполним группировку без использования ключевого слова ALL, но с вертикальной фильтрацией (с помощью раздела WHERE) — в таблице «Учебный_план» посчитаем для каждого семестра количество дисциплин с нагрузкой более 60 часов:

```
SELECT Семестр, COUNT(*) AS 'Количество часов > 60'
FROM [Учебный_план]
WHERE Количество_часов > 60
GROUP BY Семестр
```

Результат запроса показан на рис. 9.26.

Семестр	Количество часов > 60
1	4
2	2
3	2
4	1
8	1

Рис. 9.26. Дисциплины с нагрузкой более 60 часов: не все семестры попали в таблицу

В полученной таблице отсутствуют данные для пятого, шестого и седьмого семестров. Это означает, что дисциплин, удовлетворяющих поставленному условию, в семестрах нет.

Добавим в раздел GROUP BY ключевое слово ALL:

```
SELECT Семестр, COUNT(*) AS 'Количество часов > 60'
FROM [Учебный_план]
WHERE Количество_часов > 60
GROUP BY ALL Семестр
```

Будет получен результат, показанный на рис. 9.27.

	Семестр	Количество часов > 60
▶	1	4
	2	2
	3	2
	4	1
	5	0
	6	0
	7	0
	8	1
	9	0
*		

Рис. 9.27. Дисциплины с нагрузкой более 60 часов: в таблице представлены все семестры

Раздел COMPUTE

Этот раздел предназначен для выполнения групповых операций над содержимым столбцов выборки. Групповые операции задаются с помощью функций агрегирования. Результат агрегирования выводится в отдельной строке после всех данных столбца.

Синтаксис раздела COMPUTE следующий:

```
COMPUTE <Функция_агрегирования>(<столбец_агрегирования>)}
[,..., n]
    [ BY <столбец_группировки> [,...,n ] ]
```

Аргумент <столбец_агрегирования> должен содержать имя агрегируемого столбца. Этот столбец должен быть включен в результат выборки. Ключевое слово BY указывает, что результат вычисления следует сгруппировать. Следующий за этим ключевым словом аргумент <столбец_группировки> содержит имя столбца, по которому будет производиться группировка. Результат необходимо предваритель-

но отсортировать по этому столбцу, то есть столбец должен быть указан в разделе ORDER BY. Приведем простой пример применения раздела COMPUTE для вычисления количества дисциплин, читаемых в семестре, и общей суммы часов:

```
SELECT Наименование, Семестр, Количество_часов
FROM Учебный_план, Дисциплины
WHERE (Учебный_план.ID_Дисциплина =
      Дисциплины.ID_Дисциплина) AND (Семестр = 2)
COMPUTE SUM(Количество_часов), COUNT(Семестр)
```

Будет получен следующий результат:

```
Наименование                    Семестр
Количество_часов
------------------------------------ ----------
--------
Английский язык                 2          110
Физическая культура             2          130
Философия                       2          32
Экономическая теория            2          34
История мировых цивилизаций     2          20
Математический анализ           2          56
Основы алгебры и геометрии      2          52
Основы программирования         2          26

                                           sum
                                           ==========
                                           460

                                cnt
                                ==========
                                8

(9 row(s) affected)
```

Рассмотрим пример группировки при использовании раздела COMPUTE (составление списков групп и вычисление количества студентов в группе):

```
SELECT Фамилия, Имя, Отчество, Номер_Группы
FROM Студенты
ORDER BY Номер_Группы
COMPUTE COUNT(Номер_Группы) BY Номер_Группы
```

Будет получен следующий результат:

```
Фамилия      Имя         Отчество           Номер_Группы
------------ ----------- ------------------ ------------
Агапов       Иван        Иванович           2002\1
Агуреев      Дмитрий     Александрович      2002\1
Гогешвили    Серго       Тамазович          2002\1
Григорьева   Мария       Александровна      2002\1
Желтов       Олег        Игоревич           2002\1
Жуков        Виктор      Владимирович       2002\1
Жучков       Сергей      Сергеевич          2002\1
Захаркин     Николай     Владимирович       2002\1
Иванов       Олег        Геннадиевич        2002\1
Кадаков      Антон       Дмитриевич         2002\1
Леонтьев     Богдан      Вадимович          2002\1
Миняйло      Евгений     Николаевич         2002\1
Нечаева      Ольга       Николаевна         2002\1
Парфенова    Светлана    Витальевна         2002\1
Потапкин     Александр   Александрович      2002\1
Соловьева    Полина      Сергеевна          2002\1
Федянин      Александр   Алексеевич         2002\1

cnt
==========
17

Фамилия      Имя         Отчество           Номер_Группы
------------ ----------- ------------------ ------------
Алексеев     Иван        Александрович      2002\2
Белова       Ирина       Владимировна       2002\2
Бородкина    Анна        Михайловна         2002\2
Братченко    Елена       Анатольевна        2002\2
Волков       Иван        Александрович      2002\2
Гончаров     Иван        Андреевич          2002\2
Калинин      Андрей      Евгеньевич         2002\2
Кондрашкина  Ольга       Игоревна           2002\2
Ларина       Евгения     Валерьевна         2002\2
Малкова      Дарья       Дмитриевна         2002\2
Поспелов     Игорь       Григорьевич        2002\2
Тюрина       Юлия        Александровна      2002\2
Филоненко    Петр        Алексеевич         2002\2
Юртанов      Сергей      Михайлович         2002\2

cnt
==========
14

(33 row(s) affected)
```

Раздел UNION

Раздел UNION служит для объединения результатов выборки, возвращаемых двумя и более запросами.

Рассмотрим синтаксис раздела UNION:

```
<Спецификация_Запроса_1>
UNION [ALL]
<Спецификация_Запроса_2>
...
[UNION [ALL]]
<Спецификация_Запроса_ n>
```

Чтобы к результатам запросов можно было применить операцию объединения, они должны соответствовать следующим требованиям:

- запросы должны возвращать одинаковый набор столбцов (причем необходимо гарантировать одинаковый порядок следования столбцов в каждом из запросов);
- типы данных соответствующих столбцов второго и последующих запросов должны поддерживать неявное преобразование или совпадать с типом данных столбцов первого запроса;
- ни один из результатов не может быть отсортирован с помощью раздела ORDER BY (однако общий результат может быть отсортирован, как будет показано ниже).

Указание ключевого слова ALL предписывает включать в результат повторяющиеся строки. По умолчанию повторяющиеся строки в результат не включаются.

Продемонстрируем применение раздела UNION. Рассмотрим таблицы «Кадровый_состав» и «Студенты» и попробуем построить, например, общий список и учащихся, и преподавателей, номер телефона которых начинается на 120.

Сначала построим запрос для таблицы «Кадровый_состав»:

```
SELECT Фамилия, Имя, Отчество, Должность, Телефон
FROM Кадровый_состав
WHERE Телефон LIKE '120%'
```

Результат действия запроса показан на рис. 9.28.

Фамилия	Имя	Отчество	Должность	Телефон
Цветкова	Светлана	Сидоровна	Доцент	120-1716
Китов	Кирилл	Кириллович	Проф.	120-4668

Рис. 9.28. Список преподавателей, у которых номер телефона начинается на 120

Затем построим запрос для таблицы «Студенты»:

```
SELECT Фамилия, Имя, Отчество, Телефон
FROM Студенты
WHERE Телефон LIKE '120%'
```

В результате выполнения запроса получим выборку, показанную на рис. 9.29.

Фамилия	Имя	Отчество	Телефон
Барыкин	Юрий	Владимирович	120-5448
Гулько	Александр	Юрьевич	120-7787
Кессель	Глеб	Юрьевич	120-1112
Козлова	Нина	Сергеевна	120-3335

Рис. 9.29. Студенты, у которых номер телефона начинается на 120

Теперь объединим два запроса, чтобы в результате получить единую таблицу. Заметим, что столбец *Должность* отсутствует в таблице «Студенты». Чтобы в общей таблице выделить студентов, введем в запрос для таблицы «Студенты» столбец, содержащий строку — константу «Студент» для всех записей, и объединим два запроса с помощью раздела UNION:

```
SELECT Фамилия, Имя, Отчество, Должность, Телефон
FROM Кадровый_состав
WHERE Телефон LIKE '120%'
UNION
SELECT Фамилия, Имя, Отчество, Новый_столбец =
'Студент',
       Телефон
FROM Студенты
WHERE Телефон LIKE '120%'
```

После выполнения запроса получим таблицу, показанную на рис. 9.30.

Фамилия	Имя	Отчество	Должность	Телефон
Барыкин	Юрий	Владимирович	Студент	120-5448
Гулько	Александр	Юрьевич	Студент	120-7787
Кессель	Глеб	Юрьевич	Студент	120-1112
Козлова	Нина	Сергеевна	Студент	120-3335
Цветкова	Светлана	Сидоровна	Доцент	120-1716
Китов	Кирилл	Кириллович	Проф.	120-4668

Рис. 9.30. Объединение двух запросов

При объединении таблиц столбцам итогового набора данных всегда присваиваются те же имена, что были указаны в первом из объединяемых запросов.

Упорядочим полученный список по алфавиту, добавив предложение ORDER BY:

```
SELECT Фамилия, Имя, Отчество, Должность, Телефон
FROM Кадровый_состав
WHERE Телефон LIKE '120%'
UNION
SELECT Фамилия, Имя, Отчество, Новый_столбец =
'Студент',
      Телефон
FROM Студенты
WHERE Телефон LIKE '120%'
ORDER BY Фамилия
```

Результат показан на рис. 9.31.

Фамилия	Имя	Отчество	Должность	Телефон
Барыкин	Юрий	Владимирович	Студент	120-5448
Гулько	Александр	Юрьевич	Студент	120-7787
Кессель	Глеб	Юрьевич	Студент	120-1112
Китов	Кирилл	Кириллович	Проф.	120-4668
Козлова	Нина	Сергеевна	Студент	120-3335
Цветкова	Светлана	Сидоровна	Доцент	120-1716

Рис. 9.31. Список упорядочен по алфавиту

Раздел INTO. Использование команды SELECT...INTO

При указании этой конструкции результат выполнения запроса будет сохранен в новой таблице. Синтаксис раздела INTO следующий:

```
INTO <имя_новой_таблицы>
```

Аргумент <имя_новой_таблицы> определяет имя таблицы, в которую будут вставлены результаты.

При выполнении запроса SELECT...INTO автоматически создается новая таблица с нужной структурой и в нее заносится полученный набор строк. При этом в базе данных не должно существовать таблицы, имя которой совпадает с именем таблицы, указанной в команде SELECT...INTO. Если необходимо быстро создать таблицу со структурой, позволяющей сохранить результат выполнения запроса, то лучшим выходом будет использование команды SELECT...INTO.

Синтаксис команды SELECT...INTO следующий:

```
SELECT {<имя_столбца> [[AS] <псевдоним_столбца>] [, ..., n] }
INTO <имя_новой_таблицы> FROM {<имя_исходной_таблицы> [,..., n]}
```

Приведенный вариант синтаксиса далеко не исчерпывает все возможности вставки данных с помощью команды SELECT... INTO. Допускаются практически все варианты синтаксиса запроса SELECT, то есть можно выполнять группировку, сортировку, объединение и т. д.

Рассмотрим назначение аргументов команды.

<имя_столбца> [[AS] <псевдоним_столбца>]. Аргумент <имя_столбца> задает имя столбца таблицы, который будет включен в результат. Указанный столбец должен принадлежать одной из таблиц, перечисленных в списке FROM {<имя_исходной_таблицы> [,..., n]}. Если столбцы, принадлежащие разным таблицам, имеют одинаковые имена, то для столбцов необходимо использовать псевдонимы. В противном случае произойдет попытка создать таблицу со столбцами, имеющими одинаковые имена, что приведет к ошибке, и выполнение запроса будет прервано. Указание псевдонимов также обязательно для столбцов, значения в которых формируются на основе вычисления выражений (по умолчанию такие столбцы не имеют никакого имени, что недопустимо для таблицы) и когда пользователь хочет задать столбцам в создаваемой таблице новые имена (отличные от исходных). Имя псевдонима задается с помощью параметра <псевдоним_колонки>.

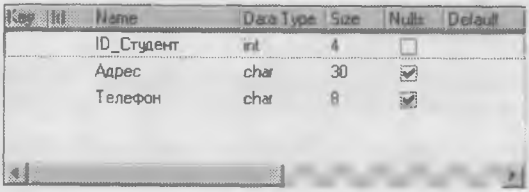
INTO <имя_новой_таблицы>. Аргумент <имя_новой_таблицы> содержит имя создаваемой таблицы. Это имя должно быть уникальным в пределах базы данных.

FROM {<имя_исходной_таблицы> [,..., n]}. В простейшем случае конструкция FROM содержит список исходных таблиц. В более сложных запросах с помощью этой конструкции определяются условия связывания двух и более таблиц.

С помощью команды SELECT...INTO, например, можно разделить таблицу «Студенты» на две, выделив в отдельную таблицу «Контакты» адреса и телефоны, а затем удалив эти столбцы из таблицы «Студенты»:

```
SELECT ID_Студент, Адрес, Телефон
INTO Контакты
FROM Студенты
```

Будет создана новая таблица, показанная на рис. 9.32.



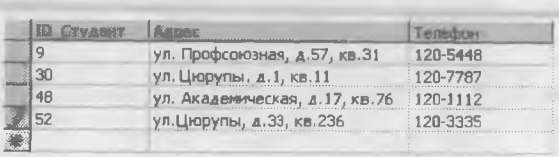
Key	Id	Name	Data Type	Size	Nulls	Default
		ID_Студент	int	4	☐	
		Адрес	char	30	☑	
		Телефон	char	8	☑	

Рис. 9.32. Структура новой таблицы

Запрос для таблицы «Контакты»:

```
SELECT *
FROM Контакты
WHERE Телефон LIKE '120%'
```

выдает результат, показанный на рис. 9.33.



ID_Студент	Адрес	Телефон
9	ул. Профсоюзная, д.57, кв.31	120-5448
30	ул. Цюрупы, д.1, кв.11	120-7787
48	ул. Академическая, д.17, кв.76	120-1112
52	ул.Цюрупы, д.33, кв.236	120-3335

Рис. 9.33. Результат запроса по номеру телефона

Построим внешний ключ для таблицы «Контакты», обеспечив связь с таблицей «Студенты»:

```
ALTER TABLE Контакты
ADD CONSTRAINT FK_Контакт
FOREIGN KEY (ID_Студент)
REFERENCES Студенты
```

Модифицируем запрос для таблицы «Контакты» (результат — на рис. 9.34):

```
SELECT *
FROM Студенты INNER JOIN
     Контакты ON
     Студенты.ID_Студент = Контакты.ID_Студент
WHERE Телефон LIKE '120%'
```

ID_Студент	Фамилия	Имя	Отчество	Номер_Группы	Год_поступления	ID_студент	Адрес	Телефон
4	Барыкин	Юрий	Владимирович	2000\2	2003	9	ул. Профсоюзная,	120-5448
30	Гулько	Александр	Юрьевич	2000\1	2003	30	ул. Цюрупы, д.1,	120-7787
48	Кессель	Глеб	Юрьевич	2000\2	2003	48	ул. Академическая	120-1112
52	Козлова	Нина	Сергеевна	2000\2	2003	52	ул. Нагорная	120-3335

Рис. 9.34. Результат запроса для таблицы «Контакты»

9.4.2. Добавление данных — команда INSERT

Рассмотрим некоторые возможности заполнения таблиц. Данные в таблицу могут быть внесены различными способами:

- с помощью команды INSERT. Используя команду INSERT, можно добавить как одну строку, так и множество строк;
- с помощью команды SELECT INTO. В этом случае на основе результата выборки, возвращаемого запросом, автоматически создается новая таблица (аппарат использования команды рассмотрен выше).

Рассмотрим процесс внесения данных в таблицу с помощью команды INSERT. Как уже было сказано, эта команда может быть использована для вставки как одной, так и множества строк.

Вставка одной строки

В простейшем случае вставка данных с помощью команды INSERT предполагает использование конструкции INSERT-VALUES:

```
INSERT [INTO] <имя_таблицы> [(<список_колонок>)]
VALUES (<список_значений>)
```

С помощью этой команды можно добавить только одну строку.

Аргумент <имя_таблицы> идентифицирует имя таблицы, в которую необходимо вставить строку данных. Необязательный параметр <список_столбцов> задает имена столбцов, в которые будет производиться добавление данных.

Рассмотрим процесс добавления данных в таблицу «Сводная_ведомость». Каждая строка этой таблицы содержит результат сдачи экзамена (зачета) по отдельной дисциплине отдельным студентом. Если студент, *ID_Студент* которого равен 10, сдал экзамен по дисциплине со значением 3 в столбце *ID_Дисциплина* на оценку «пять», то команда добавления этих данных в таблицу «Сводная_ведомость» выглядит следующим образом:

```
INSERT Сводная_ведомость
VALUES (10, 3, 5)
```

Для назначения произвольного порядка и состава столбцов в этом случае можно использовать следующую команду:

```
INSERT INTO Сводная_ведомость
(ID_Дисциплина, ID_Студент)
VALUES (3, 10)
```

Если для столбца *Оценка* определено значение по умолчанию или разрешено хранение значений NULL, то значение для этого столбца можно вообще не указывать.

Мы рассматривали вставку строк в таблицу, значения для которых были заданы с помощью констант. Однако вставляемые значения можно идентифицировать и с помощью переменных, функций, а также любых сложных выражений. Единственным требованием является совпадение типов данных столбца и значения, возвращаемого выражением.

Вставка результата запроса

Приведем упрощенный синтаксис команды INSERT:

```
INSERT [ INTO]
<имя_таблицы>
{ [ (<список_колонок>) ]
{ VALUES
( { DEFAULT | NULL | <выражение> } [, ..., n] )
| <результирующая_таблица>
}
}
| DEFAULT VALUES
```

Рассмотрим назначение каждого из аргументов.

INTO — дополнительное ключевое слово, которое может быть использовано между словом INSERT и именем таблицы для обозначения того, что следующий параметр является именем таблицы, в которую будут вставлены данные;

<имя_таблицы> — имя таблицы, в которую необходимо вставить данные;

<список_столбцов> — содержит список столбцов, в которые будет производиться вставка данных. Если он опущен, то данные будут вставляться последовательно во все столбцы, начиная с первого. Значения для столбцов указываются после ключевого слова VALUES. Для каждого столбца должно быть задано выражение, имеющее соответствующий тип данных. Если список столбцов не указан, то количество значений VALUES должно соответствовать количеству столбцов таблицы. Если же список столбцов явно задан, то это определяет порядок значений VALUES (и, соответственно, их типы). Можно не указывать явно значения для столбцов, если для них определено значение по умолчанию или разрешено хранение значений NULL.

VALUES ({ DEFAULT | NULL | <выражение> } [,..., n]) — определяет набор данных, которые будут вставлены в таблицу. Количество аргументов VALUES определяется количеством столбцов в таблице или количеством столбцов в списке (если таковой имеется). Для каждого столбца таблицы можно указать один из трех возможных вариантов:

DEFAULT — будет вставлено значение по умолчанию, определенное для столбца. Если для столбца разрешено хранение значений NULL, а значение по умолчанию не определено, то в столбец будет вставлено значение NULL.

NULL — в столбец будет вставлено значение NULL. Естественно, вставка таких значений будет успешной, если для столбца была разрешена возможность хранения значений NULL. Следует помнить, что для столбцов, входящих в первичный ключ, возможность хранения значений NULL не предусмотрена.

<выражение> — задает значение, которое будет вставлено в столбец таблицы. Этот параметр должен иметь тот же тип данных, что и столбец, а также удовлетворять ограничениям целостности, определенным для соответствующего столбца.

<результирующая_таблица> — этот параметр подразумевает указание запроса SELECT, с помощью которого будет формироваться набор данных, вставляемых в таблицу. Количество столбцов, порядок их перечисления и их типы данных должны соответствовать столбцам, указанным в списке <список_столбцов>. Если последний отсутствует, то запрос должен возвращать значения для всех столбцов таблицы.

DEFAULT VALUES — при указании этого параметра строка будет содержать только значения по умолчанию. Если для столбца не установлено значение по умолчанию, но разрешено хранение значений NULL, то в столбец будет вставлено значение NULL. Если же для столбца не разрешено хранение значений NULL, нет значения по умолчанию и в команде INSERT не указано значение для вставки, то будет выдано сообщение об ошибке и выполнение команды прервется.

Более сложный случай вставки данных предполагает использование конструкции INSERT INTO...SELECT:

```
INSERT INTO <имя_таблицы>
SELECT <выражение_запроса>
```

Аргумент <имя_таблицы> содержит имя таблицы, в которую будут вставляться выбранные данные. Таблица должна иметь соответствующую структуру и быть предварительно создана.

<Выражение_запроса> определяет тело запроса SELECT, с помощью которого производится выборка данных из одной или нескольких таблиц. Например, для выборки данных из таблицы «Студенты» обо всех студентах, поступивших в вуз в 2000 г., и сохранения их в таблице «Студент_2000» можно использовать такую последовательность инструкций:

```
CREATE TABLE Студент_2000
(ID_Студент_2000 INTEGER NOT NULL,
Фамилия      CHAR(30) NOT NULL,
Имя          CHAR(15) NOT NULL,
Отчество     CHAR(20) NOT NULL,
Адрес        CHAR(30),
Телефон      CHAR(8),
PRIMARY KEY      (ID_Студент_2000))

INSERT INTO Студент_2000
SELECT ID_Студент, Фамилия, Имя, Отчество, Адрес,
Телефон
FROM Студенты
WHERE Год_поступления = 2000
```

После выполнения этой последовательности команд инициируем запрос на отбор строк из новой таблицы:

```
SELECT TOP 5 Фамилия, Имя, Отчество
FROM Студент_2000
```

Будет выдан результат, показанный на рис. 9.35.

Приведенный пример иллюстрирует вставку строк данных в таблицу на основе результата выполнения запроса, обращающегося к од-

	Фамилия	Имя	Отчество
▶	Агупов	Петр	Александрович
	Акулов	Алексей	Юрьевич
	Амаев	Тамерлан	Джабраилович
	Барыкин	Юрий	Владимирович
	Басов	Константин	Викторович
*			

Рис. 9.35. Первые пять строк из новой таблицы «Студенты-2000»

ной таблице. Более сложные запросы могут обращаться к множеству таблиц одной или нескольких баз данных.

В качестве еще одного примера рассмотрим помещение в новую таблицу «Преподаватель-дисциплина» информации о том, какой преподаватель какую дисциплину ведет.

Для этого мы будем работать с тремя таблицами: «Кадровый_состав», «Учебный_план» и «Дисциплины». В первой таблице содержится список преподавателей, тогда как в третьей — список дисциплин. С помощью таблицы «Учебный_план» устанавливается связь «многие ко многим» между таблицами «Кадровый_состав» и «Дисциплины».

Прежде чем приступать к вставке данных, необходимо создать таблицу, которая будет содержать интересующие нас данные. Помимо столбцов для хранения информации об имени и фамилии преподавателя и названии дисциплины, предусмотрим столбцы для хранения идентификационных номеров преподавателей и дисциплин:

```
CREATE TABLE Преподаватель_дисциплина
(ID_Дисциплина    INTEGER NOT NULL,
ID_Преподаватель INTEGER NOT NULL,
Наименование   CHAR(20) NOT NULL,
Фамилия        CHAR(30) NOT NULL,
Имя            CHAR(15) NOT NULL,
Отчество     CHAR(20) NOT NULL,
Должность    CHAR(20) NOT NULL)
```

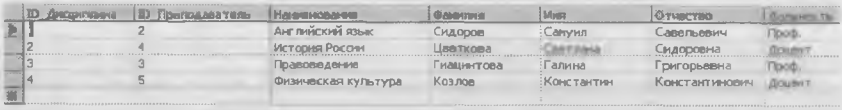
Теперь вставим в созданную таблицу нужные нам данные, выполнив для этого следующий запрос:

```
INSERT INTO Преподаватель_дисциплина
SELECT DISTINCT Дисциплины.ID_Дисциплина,
        Кадровый_состав.ID_Преподаватель, Наименование,
        Фамилия, Имя, Отчество, Должность
FROM Кадровый_состав, Учебный_план, Дисциплины
WHERE Кадровый_состав.ID_Преподаватель =
      Учебный_план.ID_Преподаватель
AND Дисциплины.ID_Дисциплина =
Учебный_план.ID_Дисциплина
```

В результате в таблицу будет вставлено 54 строки.

```
SELECT TOP 4 *
FROM Преподаватель_дисциплина
```

Результат запроса по новой таблице показан на рис. 9.36.



ID_Дисциплина	ID_Преподаватель	Наименование	Фамилия	Имя	Отчество	Должность
1	2	Английский язык	Сидоров	Самуил	Савельевич	Проф.
2	4	История России	Цветкова	Светлана	Сидоровна	Доцент
3	3	Правоведение	Гиацинтова	Галина	Григорьевна	Проф.
4	5	Физическая культура	Козлов	Константин	Константинович	Доцент

Рис. 9.36. Новая таблица «Преподаватель-дисциплина»

9.4.3. Изменение данных — команда UPDATE

Для внесения изменений в данные таблиц служит команда UPDATE, позволяющая выполнять как простое обновление данных в столбце, так и сложные операции модификации данных во множестве строк таблицы. Рассмотрим упрощенный синтаксис этой команды:

```
UPDATE <имя_таблицы>
SET { <имя_колонки> = { <выражение> | DEFAULT | NULL }}[,...,n]
{ [ FROM { <имя_исходной_таблицы> } [,...,n] ]
[ WHERE <условие_отбора> ] }
```

Рассмотрим назначение каждого из аргументов.

<имя_таблицы> — имя таблицы, в которой необходимо произвести изменение данных.

SET — с этого ключевого слова начинается блок, в котором определяется список изменяемых столбцов. За один вызов UPDATE можно изменить данные в нескольких столбцах множества строк одной таблицы.

<имя_столбца> = {<выражение> | DEFAULT | NULL} — для каждого изменяемого столбца нужно задать значение, которое он примет после выполнения изменения. С помощью ключевого слова DEFAULT можно присвоить столбцу значение, определенное для него по умолчанию. Можно также установить для столбца значение NULL. Изменению подвергнутся все строки, удовлетворяющие критериям ограничения области действия запроса UPDATE, которые задаются с помощью раздела WHERE. При составлении выражения можно ссылаться на любые столбцы таблицы, включая изменяемые. При этом следует учитывать, что изменения в данные вносятся только после выполнения команды. Таким образом, при ссылке на изменяемые столбцы будут использоваться старые значения.

FROM {<имя_исходной_таблицы>} — если при изменении данных в таблице необходимо учесть состояние данных в других таблицах, то эти источники данных необходимо указать в разделе FROM. Собственно источник данных описывается с помощью конструкции <имя_исходной_таблицы>.

WHERE <условие_отбора> — назначение раздела WHERE, используемого в запросе UPDATE, полностью соответствует назначению, которое раздел имеет в запросе SELECT, т. е. с помощью раздела WHERE можно сузить диапазон строк, в которых будет выполняться изменение данных. Необходимо указать логическое условие, на основе которого будет приниматься решение об изменении данных конкретной строки. Если в контексте значений строки указанное логическое условие выполняется (т. е. возвращает значение TRUE), то данные этой строки будут изменены. В противном случае изменение не выполняется. Предполагается, что логическое условие включает имена столбцов изменяемой таблицы, однако это необязательно.

Приведем простейший пример изменения данных. Добавим в таблицу «Учебный_план» по два часа в столбец *Количество_часов* для дисциплин 1-го семестра с формой отчетности «экзамен».

Выведем сначала исходное состояние данных (рис. 9.37):

```
SELECT *
FROM Учебный_план
WHERE (Отчетность= 'э') AND (Семестр = 1)
```

ID_План	ID_Дисциплина	Семестр	Отчетность	Количество_часов	ID_преподавателя
6	2	1	э	74	<NULL>
24	15	1	э	28	<NULL>
30	19	1	э	92	<NULL>

Рис. 9.37. Исходное состояние данных в таблице «Учебный план»

Затем выполним изменения и снова посмотрим данные (рис. 9.38).

```
UPDATE Учебный_план
     SET Количество_часов = Количество_часов + 2
     WHERE (Отчетность= 'э') AND (Семестр = 1)
SELECT *
FROM Учебный_план
WHERE (Отчетность= 'э') AND (Семестр = 1)
```

ID_План	ID_Дисциплина	Семестр	Отчетность	Количество_часов	ID_преподавателя
6	2	1	э	72	<NULL>
24	15	1	э	26	<NULL>
30	19	1	э	90	<NULL>

Рис. 9.38. В таблицу «Количество часов» внесены изменения

9.4.4. Удаление данных — команда DELETE

Удаление данных из таблицы выполняется построчно. За одну операцию можно выполнить удаление как одной строки, так и нескольких тысяч строк. Если необходимо удалить из таблицы *все* данные, то можно удалить саму таблицу. Естественно, при этом будут удалены и все хранящиеся в ней данные. Однако этот способ следует использовать лишь в самых крайних случаях, так как помимо данных будет удалена и структура таблицы.

Чаще всего удаление данных выполняется с помощью команды DELETE, удаляющей строки таблицы.

Синтаксис команды, чаще всего использующийся на практике, следующий:

```
DELETE <Имя_таблицы>
   [WHERE <Условие_отбора> ]
```

Таким образом, в большинстве случаев требуется указание лишь имени таблицы, из которой необходимо удалить данные, и логического условия, ограничивающего диапазон удаляемых строк. Причем последнее вовсе не обязательно, и при отсутствии условия из таблицы будут удалены все имеющиеся строки. Как и при выборке и изменении строк, диапазон удаляемых строк формируется с помощью раздела WHERE, использование которого было подробно рассмотрено ранее.

Пусть из таблицы «Учебный план» необходимо удалить дисциплины первого семестра с формой отчетности «зачет», т. е. строки, у которых значение в столбце *Отчетность* равно 'з'. Команда, которая позволит выполнить эту функцию, имеет следующий вид:

```
DELETE Учебный_план
     WHERE (Отчетность = 'з') AND (Семестр = 1)
```

Контрольные задания

1. Сформулируйте на языке SQL запрос для формирования экзаменационной ведомости группы студентов по дисциплине учебного плана.
2. Сформулируйте на языке SQL запрос, позволяющий сформировать листок зачетной книжки студента:

   а) по результатам сдачи экзаменов;

   б) по результатам сдачи зачетов.

3. Сформулируйте на языке SQL запрос, позволяющий получить сводную таблицу «Сессия» (см. рис. 7.2).
4. Сформулируйте на языке SQL запрос для добавления в структуру БД «Сессия» таблицы «Штатное расписание» с колонками: Должность, Разряд, Оклад, Коэффициент надбавки. Установите связь по внешнему ключу с таблицей «Кадровый состав».
5. Используя новую таблицу «Штатное расписание», сформулируйте на языке SQL запрос для расчета зарплаты с учетом коэффициента надбавки.

Глава 10
ФИЗИЧЕСКИЕ МОДЕЛИ БАЗ ДАННЫХ

Физическая модель базы данных определяет способ размещения данных на носителях (устройствах внешней памяти), а также способ и средства организации эффективного доступа к ним. Поскольку СУБД функционирует в составе и под управлением операционной системы и база данных размещается обычно на устройствах общего доступа (разделяемый ресурс), используемых самой ОС и другими прикладными программами, то организация хранения данных и доступа к ним в значительной степени зависит от принципов и методов управления данными операционной системы. И, естественно, СУБД в той или иной степени использует не только файловую систему и подсистему ввода-вывода ОС, но и специализированные методы доступа.

10.1. Организация данных на машинных носителях

С общепринятой точки зрения к вопросам организации данных относятся:

- выбор типа записи — единицы обмена в операциях ввода-вывода;
- выбор способа размещения записей в файле и, возможно, метода оптимизации размещения;
- выбор способа адресации и метода доступа к записям.

Целесообразность выделения именно таких аспектов организации была предельно очевидна на начальной стадии развития таких запись-ориентированных систем и устройств внешней памяти, как магнитные ленты и диски. Но следует отметить, что широкое использование современных поток-ориентированных систем ввода-вывода не уменьшило принципиальное, да и практическое значение давно известных методов и решений, построенных на запись-ориентированных принципах. Основные понятия и подходы к физической органи-

Рис. 10.1. Способы организации файлов данных

зации и обработке данных, кратко обсуждаемые ниже, иллюстрируются на рис. 10.1.

10.1.1. Типы записей

Логическая запись, с которой работает прикладная программа — это совокупность элементов и/или агрегатов данных, воспринимаемая и обычно физически размещаемая в рабочей области памяти прикладной программой как единое целое. Последовательность записей *в логике обработки* образует *файл*.

Физическая запись, обрабатываемая файловой системой, — это совокупность данных, которые размещаются в файле обычно на внешнем носителе и могут быть считаны или записаны как единое целое одной *командой ввода-вывода*. Здесь файл[1] — это последовательность физических записей, размещаемых в линейном пространстве носителя, но в общем случае не обязательно в линейном порядке.

[1] В некоторых операционных системах (например, фирмы IBM) файл на внешних носителях называют *набором данных* в отличие от логического файла.

Организация данных в случаях логического и физического представления может не совпадать, в частности одна физическая запись может включать несколько логических (*блокирование записей*). При этом алгоритмы выделения логических записей из физической в значительной степени зависят от *типа записи*, рассматриваемого как характер организации последовательности байтов.

На логическом уровне выделяют следующие типы:

- *записи фиксированной длины*, для размещения каждой из которых выделяется всегда память фиксированной длины, объявляемой заранее. В этом случае данные, образующие запись, имеют устойчивую природу и представляются жесткими структурами, например ряд числовых полей или символьная последовательность заданной длины;
- *записи переменной длины*, когда каждая запись может иметь длину, отличную от длины другой записи в том же наборе (например, текстовые строки), либо переменное число элементов фиксированной длины.

При этом структура представления логической записи *переменной длины*[1] отличается тем, что байтам содержания — собственно данным, образующим логическую запись, предшествуют байты значения длины содержания этой логической записи.

Существует и другая физическая структура представления записей, имеющих переменную длину, — *запись неопределенной длины*, когда данные, образующие логическую запись, завершаются разделителем «конец записи»[2].

Порядок доступа к записи неопределенной или переменной длины может быть только последовательным, поскольку для определения начала следующей записи надо считать текущую (или значение ее длины).

Для файлов записей фиксированной длины доступ будет проще, так как адрес начала любой записи может быть вычислен умножением относительного номера нужной записи на длину записи.

Физические записи на носителе следуют непосредственно друг за другом. При этом выделение отдельной записи может производиться

[1] В современных файловых системах практически не используется.

[2] В поток-ориентированных файловых системах этому соответствует организация текстовых файлов, где запись — это последовательность символов, образующих строку, которая завершается специальными кодами «CR» «LF».

двумя способами, определяемыми технологиями записи данных на носитель.

Первый способ, применяемый в запись-ориентированных устройствах внешней памяти мэйнфреймов, основан на том, что каждая запись отделяется от соседней физическим промежутком, не используемым для записи и воспринимаемым устройством чтения как сигнал «конец записи».

Другой способ — это размещение байтов следующей записи непосредственно за последним байтом предыдущей записи без каких-либо разделителей. Для этого способа характерна меньшая зависимость от особенностей устройства: оптимизация процессов ввода-вывода, в том числе блокирование[1] записей, переносится в прикладную программу.

10.1.2. Организация файлов — способ размещения записей

Записи файла обычно располагаются на носителе последовательно в том порядке, как они создаются в прикладной программе. Однако иногда физическая последовательность размещения записей может отличаться от их логической последовательности.

Последовательность размещения физических записей, естественно, может быть только одна (если содержание логической записи сознательно не дублируется в другой форме), и она должна быть выбрана с учетом эффективности использования данных в различных приложениях.

Выбор последовательности размещения связывается с одним из следующих обстоятельств:

1. Ускорение выполнения наиболее частых операций путем размещения записей в той последовательности, которая требуется при последующей обработке.

2. Ускорение или упрощение средств адресации файла (например, средств прямой адресации или хэширования).

3. Уменьшение размера используемого индекса и сокращение таким образом времени поиска в нем.

4. Сокращение среднего времени доступа за счет размещения в наиболее доступных местах записей, к которым происходит наиболее частое обращение.

---

[1] Блокирование записей переменной и неопределенной длины в этом случае будет практически невозможно.

5. Облегчение операций включения, обновления и удаления записей в интенсивно изменяемых файлах.

Можно выделить две «чистые» стратегии определения места (адреса) для размещения записей: *последовательное* (sequential) и *произвольное* (random) размещение. В этом смысле алгоритм размещения определяет *тип организации* файла.

В первом случае каждая следующая запись будет располагаться физически следом за предыдущей. Во втором — по месту, адрес которого будет определяться в зависимости от некоторых факторов, в том числе упомянутых выше.

Хотя записи на устройствах с прямым доступом могут записываться и читаться в любой последовательности, для каждой структуры данных существует некоторая определенная последовательность, в которой записи можно читать намного быстрее, чем при других способах размещения.

Рассмотрим следующие, наиболее распространенные методы организации файлов, позволяющие оптимизировать доступ к записям (рис. 10.2).

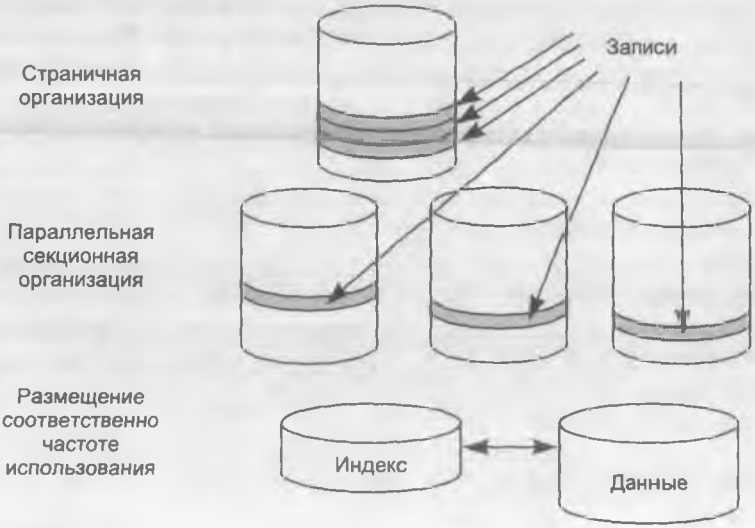


Рис. 10.2. Способы организации файлов

Страничная организация. Данные можно перемещать между внешней и оперативной памятью страницами фиксированной длины. Размер страницы определяется системой, а не длиной записи. Там, где

применяется страничная организация памяти, данные логически независимы от размера страницы, но они должны быть физически сгруппированы СУБД так, чтобы эффективно заполнять страницы.

Параллельная секционная организация. Если имеется несколько механизмов доступа, которые могут работать одновременно, то для минимизации времени ожидания данные могут быть расположены на запоминающих устройствах так, чтобы одновременно было задействовано как можно большее число механизмов доступа.

При параллельной секционной организации существуют два вида ожиданий. Запросы должны ожидать позиционирования механизма доступа (операция установки и задержки на вращение), а затем — ждать выполнения операции чтения-записи. Время, в течение которого запись читается, значительно меньше времени, в течение которого позиционируется механизм доступа. Следовательно, полное время доступа к записи при параллельной организации будет меньше.

В современных СУБД наиболее часто используется страничная организация данных, поскольку гораздо проще иметь весь файл целиком на одном пакете дисков, чем на нескольких, однако принципы секционной организации вновь нашли применение в системах планирования баз данных, а также на уровне аппаратных решений RAID-массивов.

Размещение соответственно частоте использования. Если в системах используется несколько типов запоминающих устройств или предусмотрены специальные методы доступа, то наиболее часто используемые данные можно хранить на более быстрых устройствах или в файлах с «быстрым» методом доступа.

Аналогичный принцип используется при «кэшировании», когда наиболее часто используемые записи помещаются в промежуточную память с быстрым доступом, обеспечивающимся в основном программными средствами за счет упорядочения размещения и введения избыточности.

10.1.3. Способы адресации и методы доступа к записям

Записи логического файла идентифицируются с помощью уникальной последовательности символов или некоторого числа — *ключа*. Таким ключом обычно является значение поля, расположенное в каждой записи в одной и той же позиции. Иногда бывает необходимо

объединить несколько полей, чтобы обеспечить уникальность ключа, который в этом случае называется *сцепленным ключом*.

В некоторых файлах записи имеют несколько ключей. Запись ЗАКУПКА может иметь различные НОМЕР ПОСТАВЩИКА и НОМЕР ПОКУПАТЕЛЯ, каждый из которых является ключом.

Во многих приложениях требуется идентифицировать записи по ключам, которые не являются уникальными. Однако при этом все равно должен существовать *один* уникальный ключ, тот, который используется для размещения записи в файле и выборки ее из файла. Такой ключ называется *первичным* ключом или *уникальным идентификатором*.

Основную проблему при адресации файла можно сформулировать следующим образом: *как по первичному ключу определить местоположение записи с данным ключом? Как надо организовать набор записей, чтобы поиск потребовал как можно меньше затрат?*

При разработке схем адресации файлов и определяемого ими размещения записей в файлах большое значение имеет вопрос о том, как включаются в файл новые записи и удаляются старые.

Существует несколько различных способов адресации и поиска записей, например на основе упорядочения, различных индексов, преобразования «ключ — адрес». Приведем обзор способов, количественная оценка эффективности которых представлена в [12].

Последовательное сканирование файла. Наиболее простым способом локализации записи является сканирование файла с проверкой ключа каждой записи. Этот способ, однако, требует слишком много времени и может применяться, когда каждая запись все равно должна быть прочитана.

Блочный поиск. Если записи упорядочены по ключу, то при сканировании файла не требуется чтения каждой записи. ЭВМ могла бы, например, просматривать каждую сотую запись в последовательности возрастания ключей. При нахождении записи с ключом бо́льшим, чем искомое значение, просматриваются последние 99 записей, которые были пропущены.

Этот способ называется *блочным поиском*. Записи группируются в блоки и каждый блок проверяется по одному разу до тех пор, пока не будет найден нужный блок. Иногда данный способ называют *поиском с пропусками*.

Двоичный поиск. При двоичном поиске в файле записей, упорядоченных по ключу, анализируется запись, находящаяся в середине поисковой области файла (изначально всего файла), а ее ключ сравни-

вается с поисковым ключом. Затем поисковая область делится пополам, и процесс повторяется для соответствующей половины области, пока не будет обнаружено искомое значение или длина области не станет равной 1. Число сравнений в этом случае будет меньше, чем для случая блочного поиска.

Двоичный поиск эффективен для поиска в файлах, организованных в виде двоичного дерева с указателями, когда поиск происходит в направлении, задаваемом указателями. Кроме того, добавление в файл новых записей не приводит к сдвигу других записей, что требует много времени и является достаточно сложной процедурой.

Таким образом, двоичный поиск более пригоден для поиска в индексе файла, чем в самом файле.

Индексно-последовательные файлы. Если файл упорядочен по ключам, то для адресации можно использовать *индекс,* связывающий ключ хранимой записи с ее относительным или абсолютным адресом во внешней памяти.

Индекс можно определить как структуру, с которой связана процедура, воспринимающая на входе информацию о некоторых значениях атрибутов и выдающая на выходе информацию, способствующую быстрой локализации записи или записей, которые имеют заданные значения атрибутов.

Если записи файла упорядочены по ключу, индекс обычно содержит не ссылки на каждую запись, а ссылки на блоки записей, внутри которых можно выполнять поиск или сканирование. Хранение ссылок на блоки записей, а не на отдельные записи в значительной степени уменьшает размер индекса. Однако даже в этом случае индекс часто оказывается слишком большим для поиска, и поэтому используется индекс индекса.

Индексно-произвольные файлы. Произвольный (не упорядоченный по ключу) файл можно индексировать точно так же, как и последовательный файл. Однако при этом индекс должен содержать по одному элементу для каждой записи файла, а не для блока записей. Более того, в нем должны содержаться *полные* абсолютные (или относительные) адреса, в то время как в индексе последовательного файла адреса могут содержаться в усеченном виде, так как старшие значения последовательных адресов будут совпадать.

Произвольные файлы в основном используются для обеспечения возможности адресации записей файла с несколькими ключами. Если файл упорядочен по одному ключу, то он не упорядочен по другому ключу. Для каждого типа ключей может существовать свой индекс:

для упорядоченных ключей индекс будет иметь по одному элементу на блок записей, для других ключей индексы будут более длинными, так как должны будут содержать по одному элементу для каждой записи. Ключ, который чаще всего используется при адресации файла, обычно служит для его упорядочения.

В индексно-произвольных файлах часто используются символические, а не абсолютные адреса, так как при добавлении новых или удалении старых записей изменяется местоположение записей. Если в записях имеется несколько ключей, то индекс вторичного ключа может содержать в качестве выхода первичный ключ записи. При определении же местоположения записи по ее первичному ключу можно использовать какой-нибудь другой способ адресации. По этому методу поиск осуществляется медленнее, чем поиск, при котором физический адрес записи определяется по индексу. В файлах, в которых положение записей часто изменяется, символическая адресация может оказаться предпочтительнее.

Адресация с помощью ключей, преобразуемых в адрес. Известно много методов преобразования ключа непосредственно в значение адреса в файле. В тех случаях, когда возможно преобразование значения ключа непосредственно в значение адреса в файле, такой способ адресации обеспечивает самый быстрый доступ; при этом нет необходимости организовывать поиск внутри файла или выполнять операции с индексами.

В некоторых приложениях адрес может быть вычислен на основе значений элементов данных записи.

К недостаткам данного способа относится малое заполнение файла: в файле остаются свободные участки, поскольку ключи преобразуются не в непрерывное множество адресов.

Другим недостатком схем прямой адресации является их малая гибкость. Машинные адреса записей могут измениться при обновлении файла. Для устранения этого недостатка прямую адресацию обычно выполняют в два этапа. Сначала ключ преобразуется в *порядковый номер*, который затем преобразуется в машинный адрес.

Хэширование. Простым и полезным способом вычисления адреса является хэширование (*перемешивание*). В данном методе ключ преобразуется в квазислучайное число, которое используется для определения местоположения записи.

Более экономичным является указание на область, в которой размещается группа записей. Эта область называется *участком записей* (slot, bucket).

При первоначальной загрузке файла адрес, по которому должна быть размещена запись, определяется следующим образом:

1. Ключ записи преобразуется в квазислучайное число, находящееся в диапазоне от 1 до числа участков, используемых для размещения записей.

2. Число преобразуется в адрес участка, и если на участке есть свободное место, то логическая запись размещается на нем.

3. Если участок заполнен, запись должна быть размещена на *участке переполнения* — следующий по порядку участок либо участок в отдельной *области переполнения*.

При чтении записей из файла их поиск выполняется аналогично, причем может оказаться, что для поиска записи потребуется чтение нескольких участков переполнения.

Из-за вероятностной природы алгоритма в этом способе не удается достичь 100 % плотности заполнения памяти, однако для большинства файлов может быть достигнута плотность 80 или 90 %; при этом память для индексов не требуется. Большинство записей можно найти за одно обращение, но для некоторых потребуется второе обращение (при переполнении). Для очень небольшой части записей потребуется третье или четвертое обращение к файлу.

Память в этом случае используется менее эффективно, чем в индексных методах, так как записи не упорядочены для последовательной обработки.

Комбинации способов адресации. При адресации записей используются и комбинации перечисленных выше способов. Например, с помощью индекса может определяться ограниченная поисковая область файла, затем эта область просматривается последовательно либо в ней выполняется двоичный поиск. С помощью алгоритма прямой адресации может определяться нужный раздел индекса, и, таким образом, исчезает необходимость поиска во всем индексе.

10.1.4. Схемы организации данных на внешних носителях

Схема адресации записей в файле является определяющей для способов размещения записей, т. е. условий и процедур включения в файл новых записей, обновления и удаления старых.

Записи располагаются на внешнем запоминающем устройстве в конкретной физической последовательности. Обработка данных усложняется, если последовательность записей файла не соответствует

последовательности их обработки: возникает необходимость сортировки записей, что требует значительных временных затрат. Как отмечалось ранее, другой способ организации доступа к записям в нужной последовательности, отличной от порядка физического размещения, — это использование различных систем адресации.

Для иллюстрации взаимосвязи схем адресации и организации наборов данных рассмотрим процедуру ведения массива индексно-последовательной организации.

При индексно-последовательной организации записи физически размещаются последовательно (или произвольно для индексно-произвольной организации) в соответствии с возрастанием их ключей, которые чаще всего используются для адресации этих записей.

Для работы с индексно-последовательным файлом можно использовать два режима обработки: 1) *последовательную обработку*, при которой записи обрабатываются в последовательности их размещения на внешнем запоминающем устройстве, и 2) *произвольную обработку*, при которой записи обрабатываются в произвольной последовательности, не связанной с физической организацией записей на внешнем устройстве.

На рис. 10.3 приведена схема индексно-последовательного файла, в который добавлены три новые записи.

Новые записи просто включаются в конец файла, и при этом не требуются указатели на область переполнения и выполнение специальных программ поддержки включения записей. Однако в данном случае возникает необходимость перегруппировки элементов индекса.

Существует три *способа адресации записей в области переполнения*. В первом методе применяются указатели, расположенные в индексах и указывающие на записи, содержащиеся в области переполнения. Кроме этого, в самой области переполнения используются указатели, связывающие записи в цепочки в порядке возрастания ключа.

Второй способ адресации отличается от первого лишь тем, что указатели на область переполнения создаются не для цепочек записей, а для каждой записи переполнения. В этом случае в индексе резервируется место для нескольких указателей. При этом существенно усложняется ведение индексов, а также увеличивается объем памяти для индексов (в связи с увеличением числа указателей).

Третий способ также позволяет избежать поиска записей по цепочкам благодаря созданию специального индекса независимой области переполнения. В этом случае для поиска записи, находящейся в области переполнения, необходимо прочитать индекс независимой

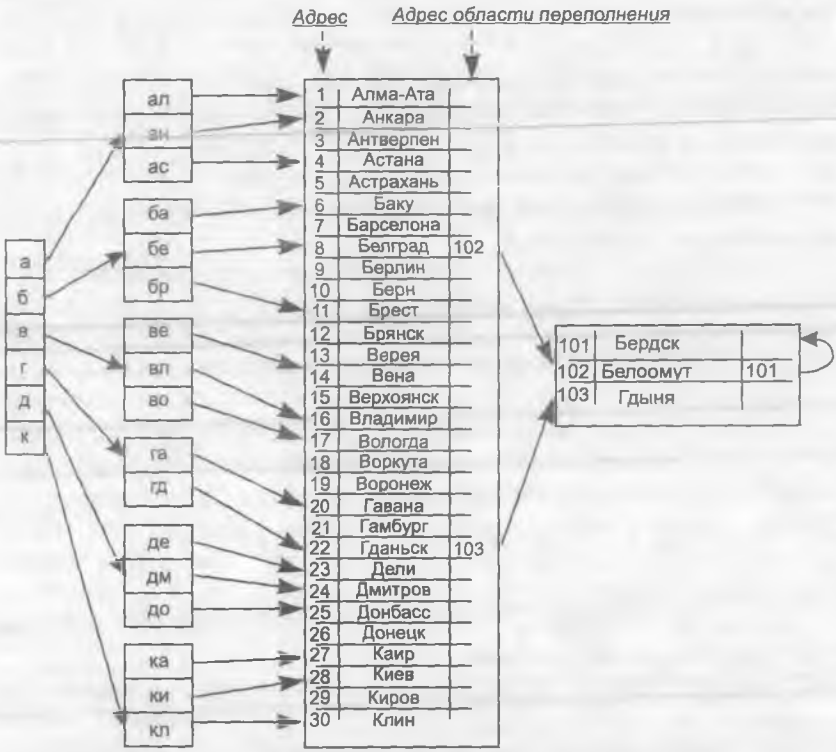


Рис. 10.3. Схема индексно-последовательного файла после добавления записей

области переполнения и считать соответствующую запись. Данный способ требует бóльших затрат времени по сравнению с первым способом, когда записи переполнения связаны в цепочки; но при многократном включении групп записей опасность возникновения очень длинных цепочек здесь отсутствует.

Методы включения записей, основанные на резервировании. Метод, основанный на резервировании участков пространства (например, фиксированной части каждого блока) в файле для ожидаемого включения новых элементов данных, называется *методом распределенной свободной памяти*. Применяя данный метод, можно избежать записей переполнения, однако целесообразно периодически выполнять процедуры восстановления заполненных резервных позиций.

Наличие некоторого объема свободной памяти в каждом управляемом интервале приводит к тому, что бóльшая часть вновь посту-

пающих записей умещается в пределах соответствующих интервалов. Тем не менее неизбежны случаи нехватки распределенной свободной памяти в интервалах для включения новых записей. В таких случаях осуществляется *«расщепление» интервала*. Предположим, что необходимо включить запись с некоторым значением ключевого поля. В соответствии со значением ключевого поля определяется интервал, в который следует включить запись. Однако интервал уже полностью заполнен, поэтому осуществляется его расщепление: половина его записей пересылается в свободный интервал, входящий в состав той же управляемой области.

Резюмируя, перечислим способы включения в файл новых записей.

1. При включении новых записей файл перезаписывается с размещением записей в соответствующих местах.

2. Записи размещаются в области переполнения, которая находится либо в той же области (файле), что и основная область, либо в отдельной независимой области (файле). При этом для обеспечения доступа могут использоваться цепочки, указатели из индекса к каждой записи переполнения, отдельные индексы для каждого блока области переполнения.

3. Записи размещаются в распределенной свободной памяти, которая резервируется на уровне физических или логических областей в пространстве файла. При переполнении первоначально зарезервированной области производится ее расщепление.

10.2. Структуры данных

Задачи, в которых точные объемы обрабатываемых данных заранее неизвестны, для своего решения требуют использования структур данных, обладающих способностью к изменению в процессе обработки.

Рассмотрим типологию и разновидности таких структур данных с точки зрения особенности их организации. Структуру данных определяют:

- алгоритм выборки отдельных элементов данных;
- особенности организации и обработки информации.

Таким образом, структура данных — это способ отображения значений в памяти: размер области и порядок ее выделения (который и определит характер процедуры адресации/выборки). Зачастую именно успешность структурирования данных определяет сложность процедур их обработки [2].

Классификация структур данных должна проводиться с двух точек зрения.

1) по характеру взаимосвязи элементов структуры (с точки зрения порядка их размещения/выборки) структуры можно разделить на линейные и нелинейные;

2) по характеру информации, предоставляемой структурой (с точки зрения однородности и «элементарности» типов данных) — на однородные структуры, где все элементы имеют один тип данных, и неоднородные (композиционные), где элементы относятся к разным типам данных.

10.2.1. Линейные структуры данных

К линейным структурам традиционно относят последовательности. Порядок следования (и, соответственно, выборки) компонентов таких структур имеет линейный характер и соответствует порядку их расположения в памяти: один за другим без каких-либо промежутков. Адрес компонента соответствует его положению и определяется индексом, задающим порядковый номер компонента в последовательности размещения. К отдельному компоненту имеется прямой доступ, если известен его индекс. Индекс в этом случае позволяет достаточно просто вычислять значение физического адреса элемента по значению его индекса.

Последовательность представляет собой совокупность однотипных элементов, однако число элементов до размещения неизвестно (до начала обработки и размещения необходимо считать длину последовательности бесконечной). Принципиальность такого предположения выражается в том, что необходимо предусматривать специальную процедуру использования памяти (выделения/освобождения) и, возможно, алгоритм обработки последовательности по частям. Важность рассмотрения такого типа данных обусловлена тем, что именно он превалирует в операциях ввода/вывода с устройствами внешней памяти. Именно последовательный доступ позволяет организовать «потоковые» операции: однородность позволяет рассматривать пересылаемые данные как непрерывный поток. Поток не может быть прерван по контекстно определяемому условию, например при пересылке текста — по значению кода «перевод строки», и это не заставляет программу анализировать значение каждого очередного элемента. И, кроме того, последовательный доступ — это простота управления памятью и устройством ввода-вывода.

В зависимости от типа элементов, вида разрешенных операций и способов использования существует несколько разновидностей последовательности. Типичными примерами последовательности являются последовательности символов, последовательный файл, очередь и стек.

Основными операциями над данными последовательности являются:

1) формирование последовательности;
2) выборка элементов последовательности;
3) включение — исключение элементов в последовательность.

В зависимости от ограничений на использование перечисленных операций выделяют несколько разновидностей последовательностей. Далее рассмотрены наиболее важные из них.

Последовательный файл. Последовательный файл представляет собой последовательность элементов, в которой допускается выборка и исключение начального элемента и добавление элемента в конец последовательности (рис. 10.4). Такие последовательности реализуются на внешней запоминающей среде, причем их запись возможна только в одном направлении.

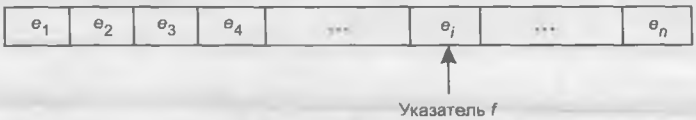


Рис. 10.4. Последовательный файл

Обработка файла реализуется путем введения файловой переменной *f* и дополнительной переменной, которая называется *буферной переменной файла* и определяет текущее значение того элемента файла, который является объектом очередной операции обработки. Совокупность значений элементов файла и положение буферной переменной определяют текущее состояние файла.

Определены следующие основные операторы, используемые для управления файлом:

- формирование нового (пустого) файла;
- установка файла в начальную позицию;
- запись в конец файла значения переменной *e;*
- считывание соответствующего положению указателя значения элемента файла в переменную *e*;
- отражение факта достижения конца файла.

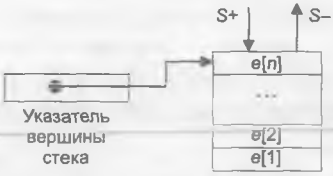


Рис. 10.5. Стековая структура

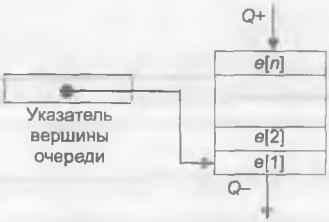


Рис. 10.6. Структура типа «очередь»

Стек. Стек является наиболее широко используемым типом данных и применяется, например, при анализе языковых конструкций (рис. 10.5). Для стека добавление и извлечение элементов возможно только с одного конца последовательности — элемент данных, добавленный к последовательности последним, извлекается из нее первым. В этом смысле стековая память является памятью с дисциплиной обслуживания «последним вошел — первым вышел» (Last In First Out — LIFO).

Очередь. Добавление элемента к очереди осуществляется с ее конца, а извлечение элемента из очереди производится с ее начала (рис. 10.6), в связи с этим очередь является памятью с дисциплиной обслуживания «первым вошел — первым вышел» (First In First Out — FIFO).

Очереди используются в случае, когда данные обрабатываются в порядке их поступления или образования.

10.2.2. Нелинейные структуры данных

В качестве примеров нелинейных структур рассмотрим списки, деревья и графы.

Порядок следования (и, соответственно, выборки) элементов таких структур может не соответствовать порядку расположения элементов в памяти. Списки представляют собой пример линейного упорядочения, деревья — двумерного, сети — произвольного. Соответственно различаются методы и средства, обеспечивающие последовательность выборки элементов данных.

Списки

Список представляет собой совокупность однотипных элементов. Однако порядок выборки элементов может отличаться от порядка следования в памяти, определенного при размещении. Наиболее очевидный способ установления однонаправленного порядка выборки элементов — это сопоставить каждому элементу списка ссылку, ука-

зывающую на следующий элемент. Для организации двунаправленного списка, допускающего также выборку в обратном порядке, каждый элемент должен иметь ссылку и на предыдущий. Такая организация уже не допускает возможности прямого доступа, например по номеру элемента.

Число элементов списка, как и в случае последовательностей, может быть неизвестно до размещения, и до начала обработки необходимо считать длину списка бесконечной, что требует наличия специальных процедур выделения/освобождения памяти.

Таким образом, с точки зрения физической реализации элемент списка должен быть *составным*, включающим собственно информативные данные и дополнительные данные (*ссылки*), определяющие порядок доступа к элементам.

В зависимости от способа построения списка и предполагаемых путей доступа к элементам различают следующие виды списков:

- однонаправленные;
- двунаправленные;
- циклические.

Наиболее простым и естественным типом списка является однонаправленный список, в котором каждый элемент содержит обязательно только одну ссылку — на следующий по порядку элемент (рис. 10.7, *а*).

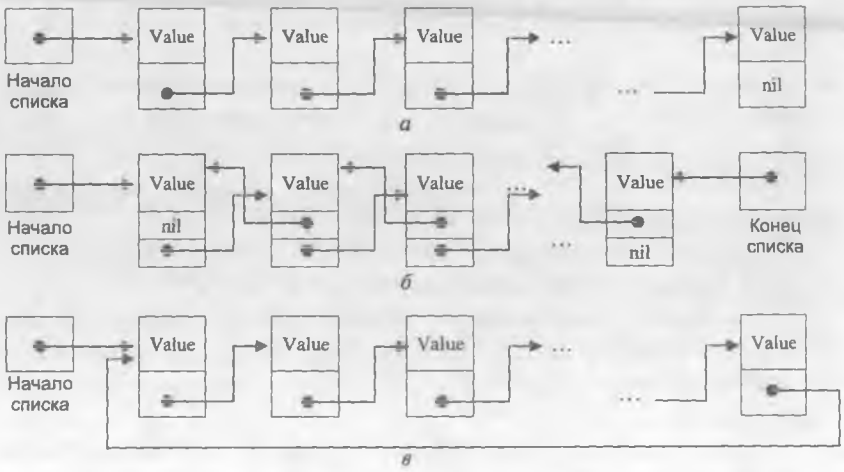


Рис. 10.7. Списковые структуры:
а — однонаправленный список; *б* — двунаправленный список; *в* — циклический однонаправленный список

Однонаправленные списки предусматривают жесткий порядок перебора элементов — только в одном направлении, от первого к последнему. Двунаправленный список представляет собой цепочку элементов, в которой каждый элемент содержит ссылку не только на следующий, но и на предыдущий (рис. 10.7, *б*).

В циклических, или кольцевых, списках порядок следования элементов зацикливается следующим образом: в однонаправленном кольцевом списке последний элемент ссылается на первый как на следующий, а в двунаправленном кольцевом списке — последний ссылается на первый как на следующий, а первый ссылается на последний как на предыдущий (рис. 10.7, *в*).

Добавление и исключение элементов списка можно выполнять с помощью простой операции изменения значений указателей.

Деревья

Дерево представляет собой иерархию элементов, называемых узлами. На самом верхнем уровне иерархии имеется только один узел — корень. Каждый узел, кроме корня, связан с одним узлом на более высоком уровне, называемым исходным узлом для данного узла. Каждый элемент имеет только один исходный. Каждый элемент может быть связан на более низком уровне с одним или несколькими элементами, которые называются порожденными. Элементы, расположенные в конце ветви, т. е. не имеющие порожденных, называются листьями.

Существует несколько способов представления структуры дерева. Например, дерево может быть определено как иерархия узлов, в которой:

1) самый верхний уровень иерархии имеет один узел, называемый *корнем*;

2) все узлы, кроме корня, связываются с одним и только одним узлом на более высоком уровне по отношению к ним самим.

Такое определение в части организации связей совпадает со списком, и, в частности, список представляет собой вырожденный случай дерева, в котором каждая вершина имеет не более одного поддерева.

В соответствии со структурой заполнения деревья подразделяются на сбалансированные и несбалансированные.

Сбалансированное дерево (в отличие от несбалансированного) в каждом узле имеет одинаковое число ветвей, причем процесс включе-

ния новых ветвей в узлы дерева идет сверху вниз, а на каждом уровне дерева — слева направо (рис. 10.8).

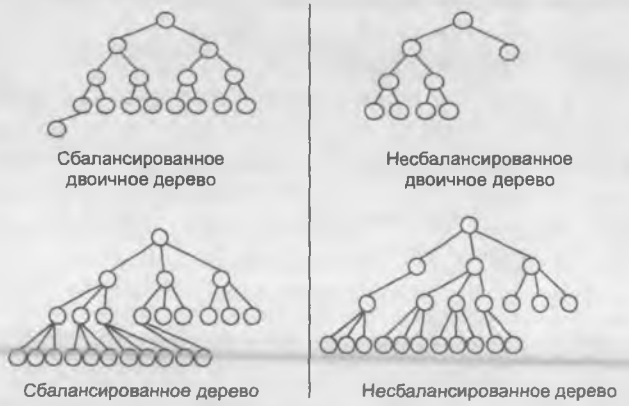


Рис. 10.8. Примеры сбалансированных и несбалансированных деревьев

Среди древовидных структур выделяют двоичные деревья. *Двоичные деревья* — это особая категория древовидных структур, в которой допускается не более двух ветвей для одного узла.

Любые связи в дереве с любым количеством ветвей можно представить в виде двоичных древовидных структур. Рисунок 10.9 иллюстрирует представление дерева в виде двоичного дерева.

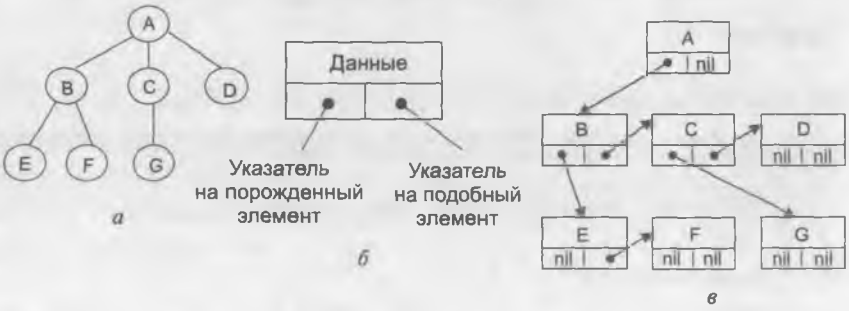


Рис. 10.9. Представление древовидной структуры в виде двоичного дерева со ссылками на подобный и порожденный элементы:
а — дерево; *б* — структура элемента (узла); *в* — структура с указателями

При таком представлении каждый элемент может иметь указатели как на порожденные, так и на подобные элементы.

Графовые структуры

Графовая структура представляет собой наиболее общий (произвольный) случай размещения и связей отдельных элементов в памяти. Рассмотренные выше списковые структуры и деревья можно рассматривать как частные случаи графа.

Существуют различные способы представления графовых структур в памяти ЭВМ. Один из них — представление графа в виде совокупности узлов и дуг. Дуги при этом представляют собой однотипные структуры, состоящие из двух частей: данные и пару указателей на левый и правый узлы. Узлы графа могут иметь структуры, различающиеся по количеству указателей (связей) (рис. 10.10).

а *б*

Рис. 10.10. Структура представления узлов и дуг графа:
а — узлы; *б* — дуги

10.3. Физическое представление иерархических структур

Рассмотрим физическое представление древовидных структур на примере операции обновления дерева с использованием следующих методов.

1) физически последовательное размещение;

2) указатели;

3) цепи и кольца.

На рис. 10.11 и 10.12 представлен пример иерархической структуры до и после обновления.

Записи, относящиеся к разным уровням дерева, обычно рассматриваются как главные и детальные. Поэтому при реализации такого файла для любой пары уровней дерева есть возможность выбора вариантов включения сегментов нижнего уровня в сегменты верхнего

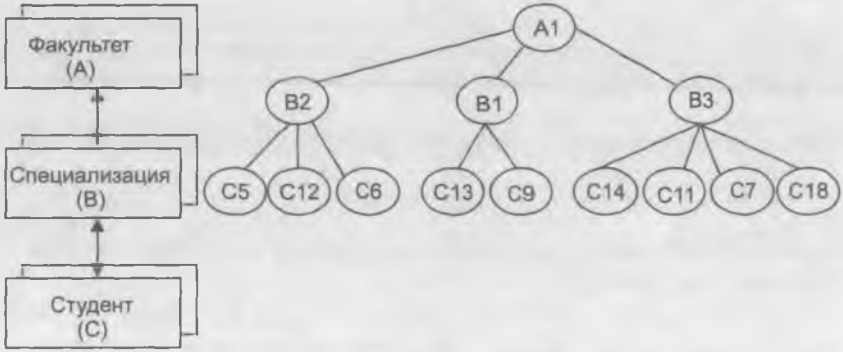


Рис. 10.11. Пример древовидной структуры

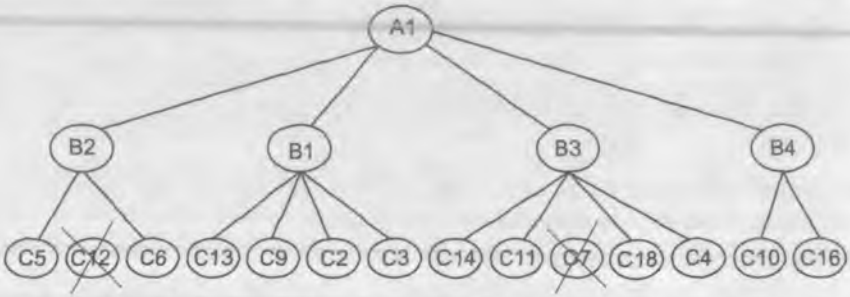


Рис. 10.12. Пример древовидной структуры после обновления

уровня. Хотя, исходя из стремления к однородности массивов, обычно все сегменты нижнего уровня размещаются отдельно от сегментов верхнего уровня.

10.3.1. Физически последовательное размещение

На рис. 10.13 и 10.14 представлен пример реализации иерархической структуры до и после обновления путем физически последовательного размещения данных на носителе.

Последовательность элементов на рис. 10.13 иногда называется *левосписковой структурой* (последовательность типа «сверху вниз — слева направо»).

Последовательность строится следующим образом: выбираются узлы, начиная от вершины дерева и вниз по самой левой ветви дерева; когда выбран узел самого нижнего уровня этой ветви, выбираются

A1	B2	C5	C12	C6	B1	C13	C9	B3	C14	C11	C7	C18

Рис. 10.13. Пример реализации древовидной структуры до обновления

A1	B2	C5	C6	B1	C13	C9	C2	C3	B3	C14	C11	C18	C4	B4	C10	C16

Рис. 10.14. Пример реализации древовидной структуры после обновления

подобные узлы слева направо; процесс повторяется, причем уже выбранные узлы пропускаются.

При размещении каждой записи последовательности в памяти может указываться, к какому уровню дерева она относится. Это выполняется путем введения в каждую запись специального кода (например, тип записи может быть определен по типу ключа). Возможно также использование некоторой формы разграничения последовательности записей, например представление записи в таком виде:

A1(B2(C5C12C6)B1(C12C9)B3(C14C11C7C18))

Последовательные левосписковые структуры не позволяют осуществлять быстрый выбор элементов нижних уровней дерева, так как при этом требуется просмотр всего списка.

10.3.2. Левосписковые структуры с переполнениями

Включение и удаление элементов могут быть выполнены с помощью метода переполнения или метода распределенной свободной памяти, рассмотренных ранее на примере метода ведения файлов с индексно-последовательной организацией данных.

На рис. 10.15 и 10.16 представлен пример реализации иерархической структуры до и после обновления путем использования области переполнения.

Рис. 10.15. Пример реализации древовидной структуры методом переполнения до обновления

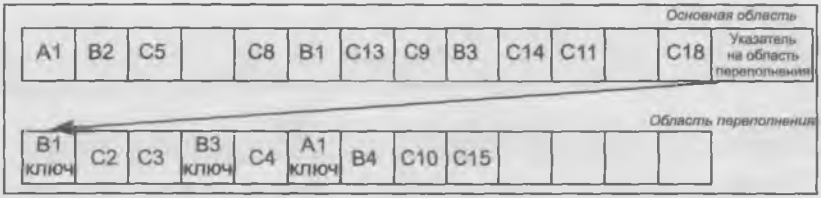


Рис. 10.16. Пример реализации древовидной структуры методом переполнения после обновления

В этом случае для определения местонахождения записей A, B или C можно использовать индексы.

10.3.3. Использование указателей на «подобные» и «порожденные»

Для обеспечения эффективных процедур выборки записей могут использоваться межзаписные ссылки следующих типов:

- указатели на порожденные записи;
- указатели на подобные записи;
- указатели на исходные записи.

При построении древовидных структур, в которых используется какой-либо один тип указателя, всегда исходят из альтернативы между сложностью реализации списка указателей переменной длины на порожденные записи и увеличением времени поиска, связанным с использованием цепочки указателей на подобные записи.

Практически эффективные компромиссы могут быть достигнуты путем использования в каждой записи двух указателей каких-либо двух типов (рис. 10.17), а также использованием кольцевых структур (например, рис. 10.18).

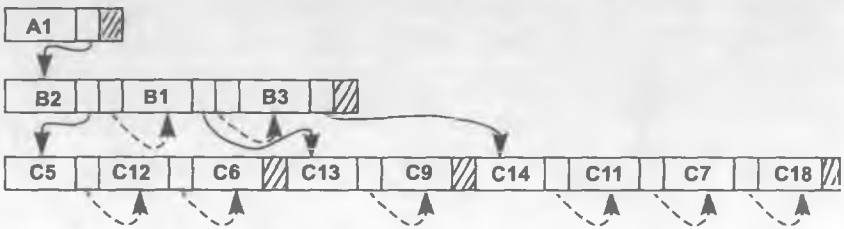


Рис. 10.17. Пример реализации древовидной структуры (см. рис. 4.5) с использованием ссылок «порожденный — подобный» (штрихованные области означают конец списка)

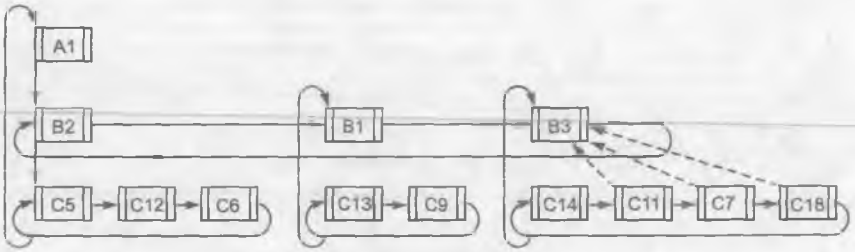


Рис. 10.18. Пример реализации древовидной структуры с использованием кольцевых ссылок

На рис. 10.18 ссылки образуют кольца двух типов: подобных записей и кольца «исходный—порожденный». В записях самого нижнего уровня показаны указатели на исходные записи. Для единообразия здесь каждая запись имеет два указателя. Однако кольца большей частью создаются двусторонними. В этом случае число указателей в каждой записи увеличится до четырех.

10.4. Физическое представление сетевых структур

Так же как и в случае древовидных структур, рассмотренных в предыдущей главе, связи в сетевых структурах можно представить,

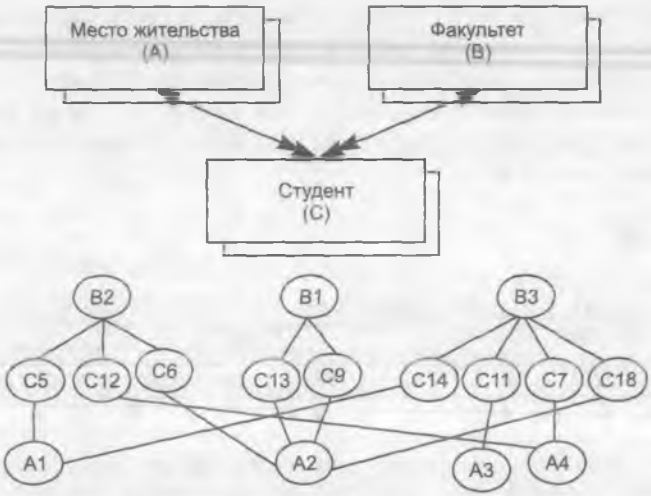


Рис. 10.19. Пример сетевой структуры

используя физически последовательное размещение, указатели, кольца. Рассмотрим простую сетевую структуру, представленную на рис. 10.19.

10.4.1. Физически последовательное размещение

Если древовидные структуры можно представить без избыточности с помощью физически последовательного размещения, то для сетевых структур это обычно невозможно. Однако в некоторых случаях может оказаться удобным представить один *набор связей* типа «исходный — порожденный» путем физически последовательного размещения, а для остальных связей использовать другой метод. Например, можно использовать физически последовательное размещение для представления связей A и C (рис. 10.20).

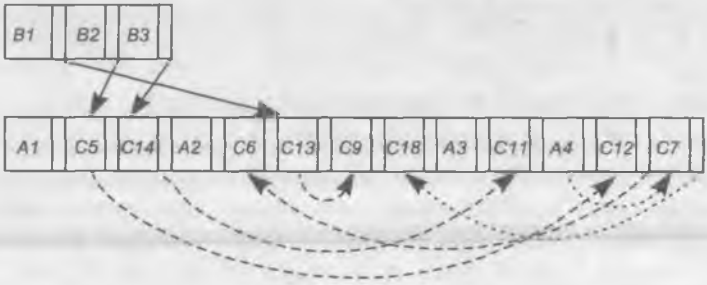


Рис. 10.20. Пример реализации сетевой структуры с последовательным размещением

В этом примере связи между B и C реализуются с помощью указателей на порожденные и подобные записи.

Обычно для представления сетевых структур физически последовательное размещение не применяется.

10.4.2. Использование указателей

Если для реализации сетевых структур используются указатели, то они должны представлять все связи, причем какие-то записи должны называться исходными (например, верхние), а какие-то — порожденными (нижние записи).

На практике могут использоваться много различных вариантов конфигураций указателей. На рис. 10.21 показаны кольцевые струк-

туры, в которых имеются указатели на исходные, порожденные и подобные записи.

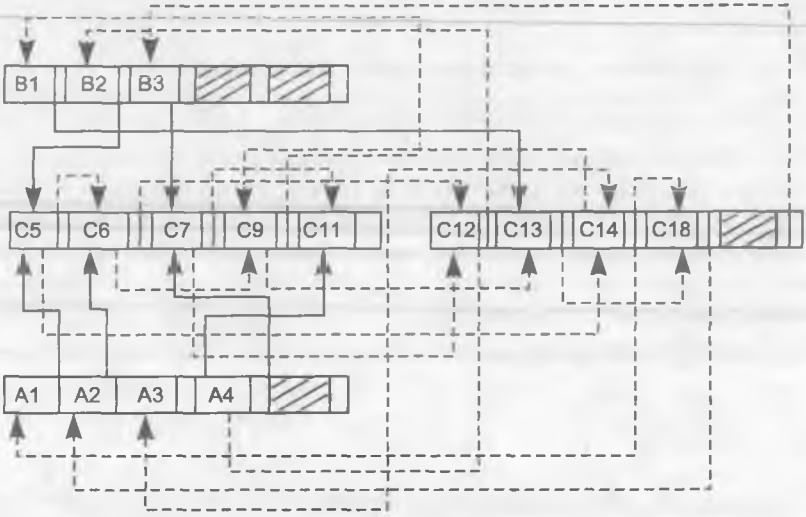


Рис. 10.21. Пример реализации сетевой структуры с использованием указателей

Однако если какая-нибудь связь между записями относится к типу «многие ко многим», то названные три метода физического представления сетевых структур оказываются непригодными. Более того, если в простых сетевых структурах на предыдущих рисунках для хранения *указателей на исходные записи* требовался один или два указателя в каждой записи, то здесь необходимы списки указателей переменной длины.

Основной проблемой, возникающей при организации встроенных списков указателей переменной длины, является сложность их ведения. При обновлении файла должна существовать возможность сокращения и удлинения списков, что обычно приводит к необходимости периодической реорганизации. Реорганизация является сложной задачей, поскольку при перемещении записей должны быть изменены многие указатели.

Эта проблема частично решается, если использовать *символические указатели,* которые не изменяются при перемещении записей. Однако их применение отражается на механизме адресации и при поиске записей в файле: на поиск записей затрачивается больше времени, чем при использовании прямых указателей.

10.5. Физическое представление с разделением данных и связей

Рассматриваемые ранее структуры в основном ориентированы на то, чтобы связи между данными хранились вместе с самими данными. Такое объединение реализовалось, например, агрегированием данных (построением сложных понятийных структур и данных) или введением ссылочного аппарата, фиксирующего семантические связи, непосредственно в записи данных.

Табличная форма представления информации является наиболее распространенной и понятной. Кроме того, такие семантически более сложные формы, как деревья и сети, путем введения некоторой избыточности могут быть сведены к табличным. При этом связи между данными также будут представлены в форме двумерных таблиц.

Такой *реляционный* подход, в основе которого лежит принцип разделения данных и связей, обеспечивает, с одной стороны, независимость данных, а с другой — более простые способы реализации хранения и обновления.

На рис. 10.23 и 10.24 приведен пример разделения линейных записей исходной таблицы «Штатное расписание факультета» (рис. 10.22) на связи и собственно данные.

В разделенном варианте получены три таблицы бинарных отношений для трех вторичных ключей и одна таблица отношений в не-

Ф. И.О.	*Год рожд.*	*Должн.*	*Каф. №*
Иванов И.И.	1948	Зав.каф.	22
Сидоров С.С.	1953	Проф.	22
Гиацинтова Г.Г.	1945	Проф.	22
Цветкова С.С.	1960	Доцент	22
Козлов К.К.	1959	Доцент	22
Петров П.П.	1960	Ст.преп.	22
Лютикова Л.Л.	1977	Ассистент	22
Рыбин Р.Р.	1950	Зав.каф.	23
Китов К.К.	1944	Проф.	23
Раков В.В.	1958	Доцент	23
Соловьева С.С.	1958	Доцент	23
Воробьева В.В.	1959	Ст. преп.	23
Орлова О.О.	1966	Ассистент	23
Осетров С.С.	1976	Ассистент	23

Рис. 10.22. Пример набора записей табличного типа

Данные

1	Воробьева В.В.
2	Гиацинтова Г.Г.
3	Иванов И.И.
4	Китов К.К.
5	Козлов К.К.
6	Лютикова Л.Л.
7	Орлова О.О.
8	Осетров С.С.
9	Петров П.П.
10	Раков В.В.
11	Рыбин Р.Р.
12	Сидоров С.С.
13	Соловьева С.С.
14	Цветкова С.С.

1	1944
2	1945
3	1948
4	1950
5	1953
6	1958
7	1959
8	1960
9	1966
10	1976
11	1977

1	Ассистент
2	Доцент
3	Зав.каф.
4	Проф.
5	Ст. преп.

1	22
2	23

Рис. 10.23. Пример разделения на связи и данные набора записей табличного типа (данные)

Связи

Ф.И.О.	Год рожд.	Должн.	Каф. №
3	3	3	1
12	5	4	1
2	2	4	1
14	8	2	1
5	7	2	1
9	8	2	1
6	11	1	1
11	4	3	2
4	1	4	2
10	6	2	2
13	6	2	2
1	7	5	2
7	9	1	2
8	10	1	2

Должн.	Ф.И.О.
1	6
1	7
1	8
2	14
2	5
2	10
2	13
3	3
3	11
4	12
4	2
4	4
5	9
5	1

Год рожд.	Ф.И.О.
1	4
2	2
3	3
4	11
5	12
6	10
6	13
7	5
7	1
8	14
8	9
9	7
10	8
11	6

Каф. №	Ф.И.О.
1	3
1	12
1	2
1	14
1	5
1	9
1	6
2	11
2	4
2	10
2	13
2	1
2	7

Рис. 10.24. Пример разделения на связи и данные набора записей табличного типа (связи)

инвертированной форме, но упорядоченная по первичному ключу. Каждое значение элемента данных представлено в одном экземпляре и имеет идентификатор (порядковый номер — ключ). Связи элементов данных также выделены в таблицы отдельно.

Такое представление обладает следующими важными свойствами:

- каждый элемент таблицы — это *один* элемент данных;
- таблица не содержит *одинаковых* строк, т. е. содержащих попарно равных значений элементов данных;
- столбцы таблицы однородны (так как элементы данных каждого столбца имеют общую природу) и могут быть однозначно идентифицированы *именованием*.

Для более сложных случаев, например древовидных структур, для устранения зависимости от путей вводятся дополнительные ключевые элементы данных.

Следует отметить, что дублирование некоторых элементов в таблицах является *логическим* и не обязательно повлечет дублирование на физическом уровне, так как можно воспользоваться указателями.

Однородность реляционных баз данных, построенных на основе бинарных отношений, обеспечивает:

- унифицированность средств работы с базой: необходимы только средства для работы с бинарными таблицами;
- простоту расширения состава логической записи.

В то же время для получения ответа по комплексному запросу необходимо обращаться к нескольким таблицам.

10.6. Архитектура файловой организации баз данных

Файловая структура и система управления файлами являются прерогативой операционной среды, поэтому по отношению к базам данных, ориентированным на работу с элементами данных и высокую интенсивность обмена, эффективность операций ввода-вывода будет не оптимальна: стандартный язык СУБД намного богаче, чем набор операций файловой системы.

Прямое использование файловой системы для организации хранения и доступа к данным оказывается менее эффективным (отличие составляет по крайней мере 10 %), поскольку при выполнении каждого обращения к диску со стороны СУБД в работу включается дополнительный слой системного ПО. Хранение данных в файловой

системе приводит также к определенной потере емкости памяти. Файловая система, например Unix, потребляет примерно 10 % от форматированной емкости дисков для метаданных о файлах и файловой системе. Более того, файловая система резервирует некоторое пространство, чтобы обеспечить быстрый поиск свободного пространства в случае расширения файлов.

Это послужило причиной того, что СУБД берут на себя непосредственное управление внешней памятью, минимально используя файловую систему ОС.

10.6.1. Файл-ориентированная организация данных

Этот подход отражает точку зрения «идейно чистого» программирования, выражающуюся в стремлении к построению модульных процедур, ориентированных на обработку регулярных однородных данных[1]: «сколько типов структур записей — столько и файлов» (рис. 10.25).

Рис. 10.25. Два подхода к организации данных

[1] Именно такой подход обеспечил возможность реализации надежных достаточно эффективных СУБД, функционирующих по современным меркам в крайне скромных рамках наличных вычислительных ресурсов.

Таким образом, БД физически состоит из нескольких файлов: основного, индексного, файла метаданных, файлов указателей и т. д. Такого типа организация файловой структуры БД представлена в приложении примерами организации данных dBase-подобных и документальных баз данных.

10.6.2. Страничная организация данных

Другой подход отражает стремление разработчиков сосредоточить в СУБД управление данными на всех уровнях — от логической обработки до управления пространством носителя. Создание сложных специализированных процедур, эффективно работающих со сложными нерегулярными структурами данных в сочетании с огромными ресурсами вычислительной мощности и оперативной памяти, позволило реализовать даже однофайловую[1] физическую структуру СУБД.

Приведем примерную логическую схему «страничной» организации хранения данных.

При распределении дискового пространства рассматривается следующая схема структуризации пространства в зависимости от типов данных.

Экстент — это непрерывная область дисковой памяти, включающая несколько страниц фиксированной длины. Новый экстент создается после заполнения предыдущего и связывается с ним ссылкой, которая располагается на последней странице экстента либо в специальной карте размещения. Учет свободных страниц ведется внутри экстента.

Каждый экстент используется для хранения одного из нескольких типов страниц: страницы данных, страницы индексов, страницы blob-объектов[2] (неструктурированных данных, например больших текстовых или двоичных данных). То есть данные на одной странице являются однородными: страница, например, может хранить только данные или только индексы.

Основной логической единицей операций обмена (ввода-вывода) является *страница данных,* хранящая данные в виде *строк* или других специализированных структур.

---

[1] Например, MS ACCESS.

[2] Для СУБД важно знать, что этот объект надо хранить целиком и что размеры этих объектов от записи к записи могут меняться, а в общем случае размер не ограничен.

Все страницы данных имеют одинаковую структуру, включающую:

- *заголовок страницы,* содержащий номер страницы, номера предыдущей и следующей страниц, сведения о свободном пространстве на странице;
- *содержание* — строки данных (последовательность кодов), каждая из которых имеет уникальный идентификатор в рамках всей базы данных, состоящий из номера страницы и номера строки на странице;
- *дескрипторы строк,* задающие смещение строки на странице и длину строки, что позволяет при переупорядочении строк на страницах не перемещать их физически, так как все манипуляции проводятся с дескрипторами.

Для организации быстрого доступа создаются *страницы индексов,* которые организованы обычно в виде B-деревьев.

10.7. Модели распределения данных по физическим носителям

Важным фактором, влияющим на производительность подсистемы ввода-вывода, является распределение данных по дискам. Даже минимальная по объему высокопроизводительная система должна иметь по крайней мере четыре диска: один для операционной системы и области подкачки (swap), один для данных, один для журнала и один для индексов.

Размещение всех данных БД на одном и том же диске почти всегда приводит к неудовлетворительной производительности. В частности, может оказаться, что процесс формирования журнала, который должен записываться синхронно, в действительности будет выполняться в режиме произвольного, а не последовательного доступа к диску. Уже только эта операция будет существенно задерживать каждую транзакцию[1] обновления базы данных.

Кроме того, выполнение запросов, выбирающих записи из таблицы данных путем последовательного сканирования индекса, будет сильно увеличивать время ожидания ввода-вывода. Обычно сканиро-

[1] Транзакция — неделимая последовательность операторов манипулирования данными. Подробнее см. гл. 12.

вание индекса выполняется последовательно, но в данном случае головка диска должна перемещаться для поиска каждой записи данных между выборками индексов. Наконец следует отметить, что объединение разных функций на одних и тех же физических ресурсах приводит к резкому увеличению времени подвода головок на диске.

Примером, иллюстрирующим подход с точки зрения практических компромиссов выбора решения, являются RAID-массивы. На рис. 10.26 приведены два варианта: RAID-0, обеспечивающий максимальную производительность при «стандартной» надежности, и RAID-1, обеспечивающий «двойную» надежность при «стандартной» производительности.

Системы управления базами данных применяют по крайней мере два *механизма для распределения данных* по дисковым накопителям. Для эффективного распределения доступа к данным многие СУБД имеют возможность осуществлять *сцепление* нескольких дисковых накопителей или файлов. Если по запросам производится произвольный доступ к данным, например если пользователи независимо запрашивают разные записи, то возможности сцепления дисков в СУБД полностью обеспечивают распределение нагрузки по доступу к множеству дисков (при достаточно равномерном заполнении пространства базы).

Если обращения по своей природе последовательны, в частности если один или несколько пользователей должны просматривать каждую строку таблицы, то больше подходит механизм *расщепления* дисков.

Главное отличие между сцеплением и расщеплением заключается в размещении смежных данных.

Когда диски сцепляются друг с другом, последовательное сканирование представляет собой тяжелую нагрузку для каждого из дисков, но эта нагрузка носит последовательный характер (только один диск участвует в обслуживании запроса).

Расщепление дисков осуществляет деление данных на меньшие порции, размещаемые на разные диски, позволяя тем самым всем дискам участвовать в обслуживании даже сравнительно небольшого запроса. В результате при использовании расщепления существенно уменьшается загрузка дисков при выполнении последовательного доступа. Основными кандидатами для расщепления являются обычно архивные и журнальные файлы, поскольку к ним всегда осуществляется последовательный доступ, что может ограничить общую производительность системы.

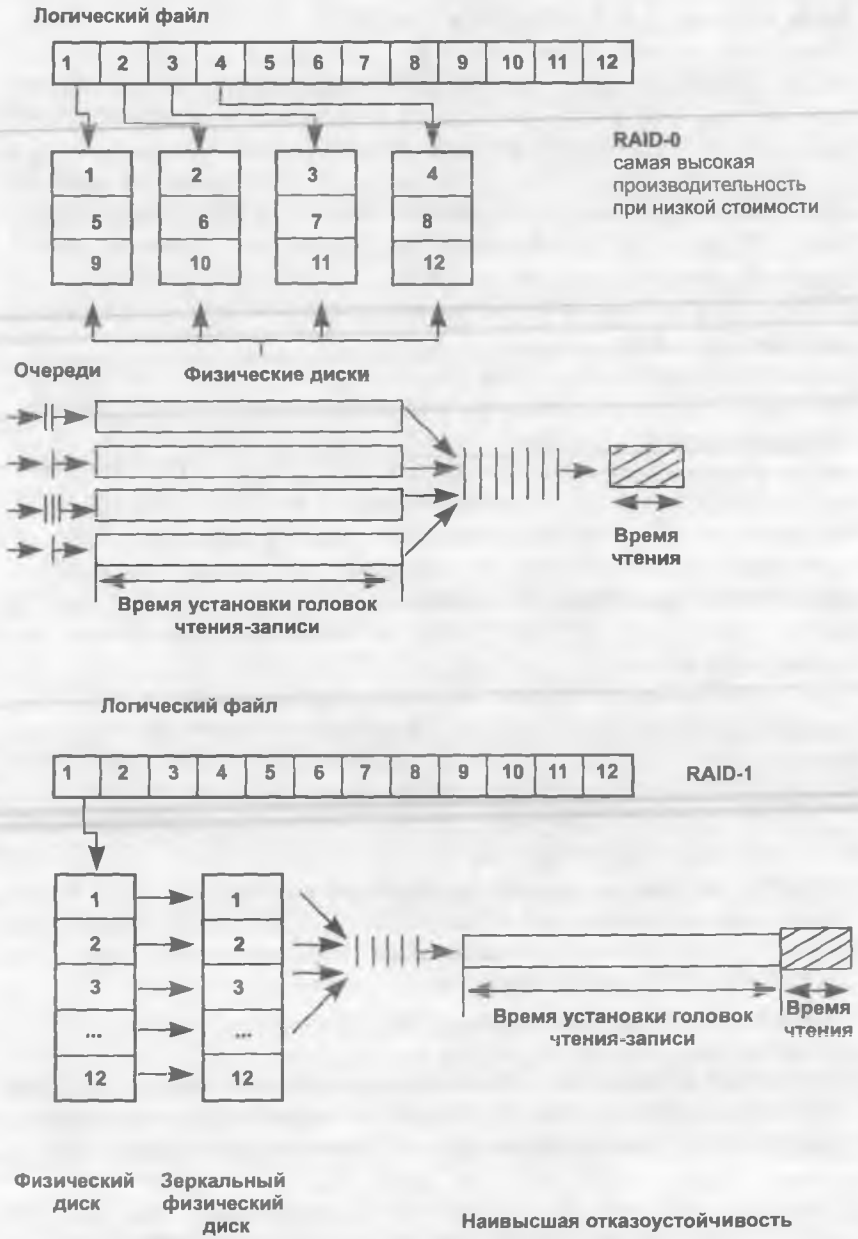


Рис. 10.26. Распределение данных в RAID-массивах

10.8. Формы организации индексов

Логически индекс есть бинарное отношение $I(v, a)$, где v — значение атрибута, а a — список адресов элементов хранения (записей, кортежей или документов), соответствующих данному значению атрибута. Для повышения эффективности поиска отдельных значений и слияния/пересечения списков a значения v и адреса элементов в списке a хранятся в упорядоченном виде, что обеспечивает ускорение поиска за счет сокращения длины перебора.

Индекс $I(v, a)$ часто называют также *инвертированным индексом* в том смысле, что значения атрибутов извлекаются из элементов хранения на поверхность, т. е. инвертируются. Каждый элемент a инвертированного индекса называют инвертированным списком.

Такое бинарное отношение на физическом уровне (имея в виду логику формирования и возможности последующего использования в процедурах поиска) может быть реализовано двумя способами — в прямой и инвертированной форме.

Прямая реализация индекса представляет собой расширение бинарного отношения до совокупности записей, состоящих из двух полей — значения атрибута и адреса размещения одного элемента хранения (рис. 10.27). При такой реализации отдельное значение атрибута повторяется в индексе столько раз, сколько оно встречается в файле данных, а длина индекса (в записях) совпадает с длиной файла данных.

Рис. 10.27. Структура индекса с прямой адресацией

В случае, когда списки *a* достаточно длинные, инвертированный индекс хранится в двух разных логических файлах, связанных указателями (файл-индекс и файл пересылок на рис. 10.28). Такая организация называется хранением индекса в инвертированной форме, требует меньше памяти для хранения значений ключей, и поиск в ней более эффективен.

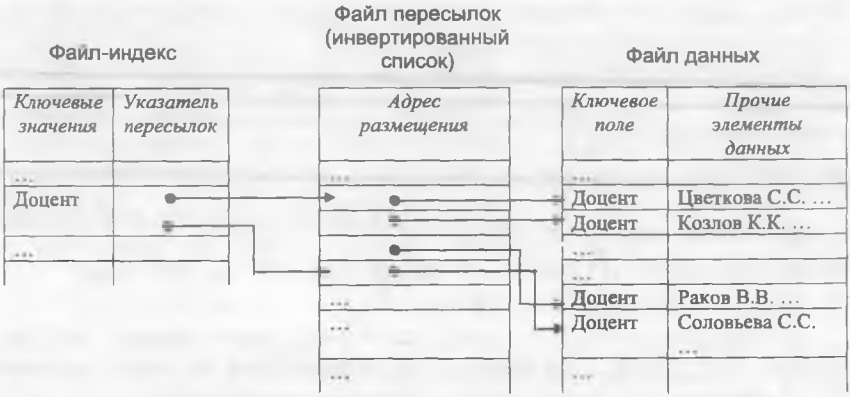


Рис. 10.28. Пример реализации инвертированного индекса

Указатель пересылок может быть реализован несколькими способами. Например,

- указанием ссылки на первый и последний адреса размещения элементов для отдельного ключевого значения, как показано на рис. 10.28;
- заданием ссылки на адрес размещения первого элемента и длиной списка адресов.

Рассмотрим далее структурные аспекты эффективных индексных организаций, ориентируясь на приведенную классификацию форм представления.

10.8.1. Типы индексов прямой формы

Индексы прямой формы структурно на физическом уровне организованы в виде сбалансированных деревьев, листья которых содержат последовательности упорядоченных пар «ключевое значение — адрес размещения».

B-дерево

Наиболее распространенная индексная структура — B-дерево, характеризующееся фиксированным размером каждого узла и переменным количеством (в зависимости от размера ключевых значений) дочерних ссылок. Такие индексы поддерживаются практически всеми СУБД и практически для всех типов данных и обеспечивают равное время поиска для каждого отдельного значения ключа. Упрощенный пример такого индекса представлен на рис. 10.29.

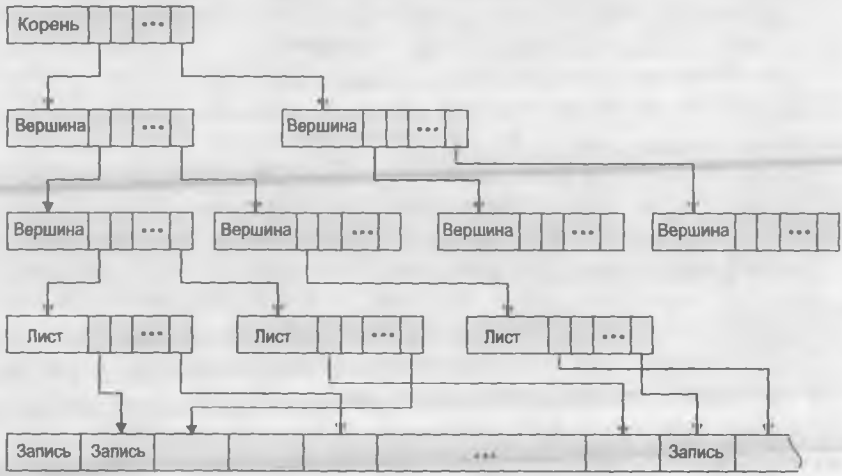


Рис. 10.29. Структура индекса (B-дерево)

Пространственные индексы

Многие СУБД (например, MySQL, PostgreSQL, MS SQL, Oracle) в настоящее время поддерживают типы данных, предназначенные для описания географических объектов (так называемые пространственные типы данных), и функции для работы с ними. Образы географических объектов представляются четырьмя основными категориями типов данных:

- *точка (point).* Точки описывают образы 0-мерных объектов и могут задавать позиции строений, статичных и передвигающихся транспортных средств и т. п.;
- *кривая (curve).* Кривые описывают такие объекты, как реки, дороги, железнодорожные линии, коммуникации коммунального хозяйства или линии разломов земной поверхности. Объекты

типа «кривая» представляются типами данных *линия* (*line*) и *последовательность линий* (*linestring*);

- *поверхность (surface)*. Поверхности могут представлять такие объекты, как страны на карте мира, города, районы, горные массивы и т. п. Поверхности описываются типом данных *многоугольник* (*polygon*);
- *геометрический набор (geometry collection)*. Геометрические наборы описывают сложные объекты, такие как группа островов, множество нефтяных скважин и т. п., и описываются типами данных *составная точка* (*multipoint*), *составная кривая* (*multicurve*) и *составная поверхность* (*multisurface*).

Для пространственных типов данных существуют особые методы индексирования, например на основе пространственных сеток (Grid-based Spatial index) и R-деревьев (R-Tree index).

Пространственная сетка (*Grid-based Spatial index*) — это древовидная структура, основанная на рекурсивном разбиении *n*-мерного пространства на сегменты равного размера (поддерживается, например, в СУБД MS SQL Server). Для двумерного случая (рис. 10.30) все представляемое пространство будет разбито на фиксированное число ячеек (например, четыре для случая Quadtree). Далее для ячеек, в которых количество объектов превышает установленный максимум, процедура разбиения повторяется, образуя следующий уровень вложенности, и т. д. Этот процесс будет продолжаться до тех пор, пока не будет достигнут максимальный порог вложенности или количество объектов в каждой ячейке не будет превышать заданного максимума.

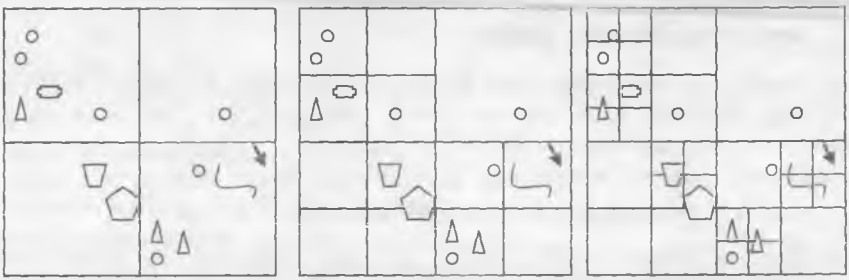


Рис. 10.30. Три уровня разбиения двумерной пространственной сетки

R-дерево (*Regions Tree*) — это сбалансированное дерево, являющееся естественным расширением B-дерева для *n* измерений. Объекты в R-дереве представлены своими минимальными ограничивающи-

ми прямоугольными параллелепипедами (поддерживается, например, в СУБД MySQL, PostgreSQL, Oracle).

Каждый не листовой узел R-дерева содержит не более чем M записей, каждая из которых задает ограничивающий прямоугольник для всех объектов дочернего узла и ссылку на соответствующий дочерний узел. Листовые узлы содержат ссылки на ограничивающие прямоугольники объектов. При этом ограничивающие прямоугольники могут пересекаться.

Стратегии разбиения узла (в случае переполнения) предполагают формирование двух дочерних узлов на основе выбора двух ограничивающих прямоугольников по критерию задаваемой ими максимальной площади или максимального расстояния. Распределение объектов узла по двум дочерним происходит по принципу минимальной совместной площади. На рис. 10.31, 10.32 представлен пример набора пространственных объектов и соответствующего ему R-дерева с максимальным числом объектов в узле, равным трем.

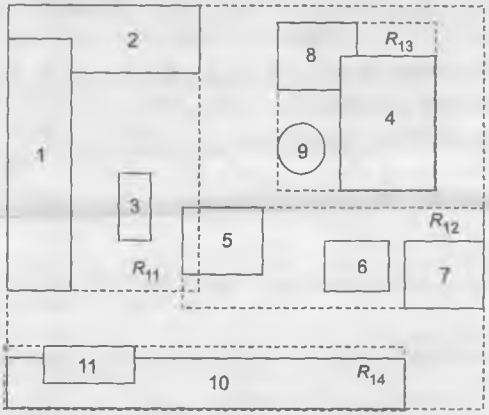


Рис. 10.31. Набор пространственных объектов

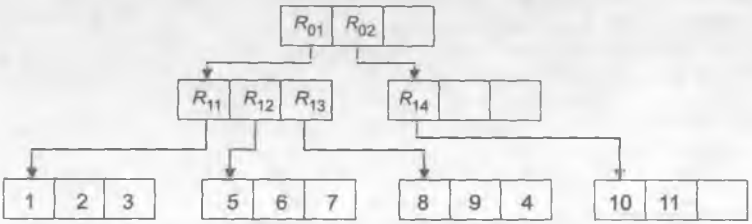


Рис. 10.32. R-дерево для пространственных объектов рис. 10.31

Индексы на основе функциональных преобразований значений атрибутов

Преобразование ключевых значений при построении индексов на основе B-дерева применяется с целью повышения эффективности поисковых операций и ввода данных в системах оперативных транзакций.

Хэш-индекс (*Hash-index*) — индекс, предполагающий хранение не самих значений ключевых атрибутов, а результатов преобразований с помощью хэш-функции. Так как хэш-функция по определению имеет ограниченное дискретное множество значений, с ее применением может быть уменьшена суммарная длина значений атрибутов, что влечет за собой сокращение размера индекса и, соответственно, повышение скорости его обработки. При запросах с использованием хэш-индексов сравниваться будут не искомые значения со значениями ключевого атрибута, а результаты применения хэш-функции (поддерживается, например, в СУБД PostgreSQL).

Индекс с реверсированным ключом (*Reversed index*) — это B-дерево, построенное на основе «перевернутых» значений ключевых атрибутов. Применяется (например, в СУБД Oracle) в основном для индексирования монотонно изменяющихся значений (например, автоинкрементных идентификаторов) в системах оперативных транзакций: последовательно вводимые данные при индексировании не будут конкурировать за последний листовой блок, а попадут в разные блоки индекса (табл. 10.1).

Таблица 10.1 **Пример значений в индексе с реверсированным ключом**

Значение атрибута в таблице		Значение атрибута в индексе	
десятичное	двоичное	десятичное	двоичное
200	11001000	19	00010011
201	11001001	147	10010011
202	11001010	83	01010011
203	11001011	211	11010011
204	11001100	51	00110011
205	11001101	179	10110011

Функциональный индекс (Function-based index) — индекс, сохраняющий в качестве ключевых значений результаты вычисления пользовательских функций (т. е., перед записью в индекс значения одного или нескольких атрибутов выступают в качестве аргументов некоторой функции). Функциональные индексы удобно строить для атрибутов, значения которых проходят предварительную обработку перед использованием в предложении WHERE SQL-запроса. Например, при сравнении строковых данных без учета регистра часто используется функция приведения текстовых строк перед сравнением к одному регистру. Создание функционального индекса с использованием функции преобразования строки к верхнему или нижнему регистру повысит эффективность таких запросов. Поддерживают функциональные индексы СУБД MS SQL Server, PostgreSQL, Oracle.

10.8.2. Типы индексов инвертированной формы

Структура индексов в инвертированной форме предполагает однократное хранение отдельного значения атрибута со ссылкой на список адресов размещения. Типы таких индексов различаются в зависимости от способа организации списка адресов размещения.

Хранение адресов размещения в виде битовых строк

В случае формирования битовых строк идентификаторы адресов размещения рассматриваются только как целые числа и представляют собой последовательные номера физических записей. Каждое отдельное значение атрибута снабжается битовым массивом, длина которого (в битах) в общем случае совпадает с количеством индексируемых записей. Таким образом, порядковый номер бита в массиве определяет соответствующий последовательный номер записи. Бит, равный 1, свидетельствует о наличии индексируемого значения атрибута в записи. Нулевой бит означает его отсутствие.

Индекс битовых карт (*Bitmap index*) формирует последовательность битов для каждого уникального значения атрибута. Эффективен в случае ограниченного числа значений индексируемого атрибута. В табл. 10.2 приведен индекс битовых карт для атрибутов *Должность* и *Кафедра* (см. рис. 10.27).

Отбор записей по критерию истинности логического выражения с использованием битовых карт сводится к побитовому выполнению логических операций. Например, реализация условия (Долж-

Таблица 10.2. **Пример индекса битовых карт**

Значение атрибута	Битовая карта													
Атрибут «Должность»														
Ассистент	0	0	0	0	0	0	1	0	0	0	0	0	1	1
Доцент	0	0	0	1	1	0	0	0	0	1	1	0	0	0
Зав. каф.	1	0	0	0	0	0	0	1	0	0	0	0	0	0
Проф.	0	1	1	0	0	0	0	0	1	0	0	0	0	0
Ст. преп.	0	0	0	0	0	1	0	0	0	0	0	1	0	0
Атрибут «Кафедра»														
22	1	1	0	0	1	1	1	0	0	1	0	0	1	0
23	0	0	1	1	0	0	0	1	1	0	1	1	0	1

ность = «Доцент») and (Кафедра = 22) приведет к логическому умножению соответствующих битовых массивов и формированию битового массива результата отбора, в котором значение 1 будет установлено в битах с номерами 5 и 10, т. е. записи с соответствующими физическими номерами будут выданы как удовлетворяющие условию отбора. Индексы битовых карт реализованы, например, в СУБД PostgreSQL и Oracle.

«*Резаный*» *битовый индекс* (*Bit-slice index*) представляет собой разновидность индекса битовых карт для значений атрибутов, тип данных которых занимает фиксированное число байтов (например, целые числа). Битовый массив в таком индексе строится для каждого отдельного бита в значении атрибута, а количество различных значений атрибута совпадает с количеством битов в типе данных атрибута. Например, если атрибут *Кафедра* определен как целое число размером в один байт, то проиндексированы будут числа от 0 до 7 (порядковые номера битов в значении атрибута), а битовый массив для каждого числа будет построен по принципу наличия или отсутствия единицы в этом бите в значении атрибута. В табл. 10.3 приведен пример такого индекса для атрибута *Кафедра* (см. рис. 10.27) и значений 22 (00010110) и 23 (00010111).

Преимущества такого представления значений атрибута — в сокращении количества битовых массивов в индексе: для целого числа,

Таблица 10.3. **Пример «резаного» битового индекса**

Номер бита в значении атрибута	Битовая карта													
	Атрибут «Кафедра»													
0	0	0	1	1	0	0	0	1	1	0	1	1	0	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1	1	1	1	1	1	1
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	1	1	1	1	1	1	1	1	1	1	1	1	1	1
5	1	1	0	0	1	1	1	0	0	1	0	0	1	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0

занимающего 1 байт, возможно 255 различных значений (от 0 до 255), в то время как в индексе их всегда будет 8 (по количеству битов в байте).

При поиске по «резаному» битовому индексу сначала по значению атрибута определяются порядковые номера битов, установленных в 1, а затем для битовых строк, соответствующих этим номерам, выполняется операция логического умножения (AND). Для примера, представленного в табл. 10.3, отбор записей со значением атрибута *Кафедра*, равным 23, необходимо выполнить операцию логического умножения строк с номерами 0, 1, 2 и 4. В результате получится битовая строка, соответствующая значению 23 в табл. 10.2. «Резаный» битовый индекс реализован, например, в СУБД Cache.

Хранение адресов размещения в виде списка

Представление ссылок на записи в виде инвертированных списков используется для так называемых *полнотекстовых индексов*, т. е. индексов, рассматривающих значение текстового атрибута как множество отдельных лексических единиц, выделяемых из текста по заложенным в программной системе правилам. После выделения лексических единиц и их упорядочивания в заданной последовательности строится индекс в инвертированной форме, т. е. отдельному

значению в индексе (отдельной лексеме) ставится в соответствие список уникальных идентификаторов записей, содержащих это значение.

№ записи	Значение атрибута
1	Автоматизированная информационная система предприятия
2	Информационная система обслуживания заказов
3	Поисковый механизм
4	Механизм обслуживания заказов

Лексическая единица	Инв. список
Автоматизированная	1
Заказов	2, 4
Информационная	1, 2
Механизм	3, 4
Обслуживания	2, 4
Поисковый	3
Предприятия	2
Система	1, 2

Рис. 10.33. Пример текстовых значений атрибута и инвертированных списков

Пример индексирования атрибута, представляющего собой текстовую строку, приведен на рис. 10.33.

Контрольные вопросы

1. Перечислите типы физических записей. Приведите примеры, показывающие соотношение физических и логических записей.
2. Перечислите факторы, влияющие на выбор метода размещения данных (организацию файла).
3. Перечислите методы организации файлов, позволяющие оптимизировать доступ к записям.
4. Приведите пример организации данных в виде индексно-последовательного файла.
5. Какие типы указателей можно использовать для реализации иерархической структуры?
6. Какие типы указателей можно использовать для реализации сетевой структуры с последовательным размещением?

7. Придумайте схему реализации сетевой структуры для случая часто изменяемых данных.
8. Какие методы организации данных, применяемые для реализации иерархических структур, неэффективны для реализации сетевых структур?
9. Приведите примерную структурную схему «страничной» организации хранения данных.
10. Приведите примерную схему размещения данных с использованием механизма расщепления.
11. Перечислите известные линейные структуры данных и приведите их характеристики.
12. Приведите основные отличия нелинейных структур данных от линейных.
13. Перечислите известные способы организации доступа к элементам списка.
14. Опишите способ представления любой древовидной структуры в виде двоичного дерева.
15. Перечислите формы организации индексов.
16. Охарактеризуйте и приведите примеры индексов прямой формы.

Глава 11
РАСПРЕДЕЛЕННАЯ ОБРАБОТКА ДАННЫХ

11.1. Основные условия и требования к распределенной обработке данных

Такая отличительная особенность БД, как многоцелевое параллельное использование данных, предопределяет наличие средств, обеспечивающих практически одновременный и независимый доступ к одним и тем же данным. Причем сама база может быть размещена на одном или нескольких компьютерах.

В [4] приводятся следующие сформулированные ведущими поставщиками СУБД свойства «идеальной» системы управления распределенными базами данных:

- *прозрачность относительно расположения данных*: все данные должны представляться так, как если бы они были локальными;
- *гетерогенность системы*: должна быть обеспечена работа с данными, которые хранятся в системах с различной архитектурой и производительностью (независимость от СУБД на отдельной ЭВМ);
- *прозрачность относительно сети*: должна быть обеспечена одинаково эффективная работа в условиях разнородных сетей;
- *поддержка распределенных запросов*: пользователь должен иметь возможность объединять данные из любых баз, даже если они размещены в разных вычислительных системах;
- *поддержка распределенных изменений*: пользователь должен иметь возможность изменять данные в любых базах, на доступ к которым у него есть права, даже если эти базы размещены в разных вычислительных системах;

- *поддержка распределенных транзакций*: должны выполняться транзакции, выходящие за рамки одной вычислительной системы, и поддерживаться целостность распределенной БД даже при возникновении отказов как в отдельных вычислительных системах, так и в сети;
- *безопасность*: должна быть обеспечена защита всей распределенной БД от несанкционированного доступа.
- *универсальность доступа*: должна использоваться единая методика доступа ко всем данным.

Однако ни одна из существующих СУБД не достигает этого идеала вследствие следующих практических проблем:

- низкая и несбалансированная производительность сетей передачи данных, что в распределенных транзакциях сильно снижает общую производительность обработки;
- обеспечение целостности данных в распределенных транзакциях базируется на принципе «все или ничего» и требует специального протокола двухфазного завершения транзакций, что приводит к длительной блокировке изменяемых данных;
- необходимость обеспечения совместимости данных стандартного типа, для хранения которых в разных системах используются разные физические форматы и кодировки;
- выбор схемы размещения системных каталогов. Если каталог будет храниться централизованно, то удаленный доступ будет замедлен. Если будет размножен — изменения придется распространять и синхронизировать;
- необходимость обеспечения совместимости СУБД разных типов и поставщиков;
- увеличение потребностей в ресурсах для координации работы приложений с целью обнаружения и устранения тупиковых ситуаций в распределенных транзакциях.

Именно указанные причины определили на практике частичность и «этапность» введения в СУБД тех или иных возможностей распределенной обработки данных. В простейшем случае пользователь имеет возможность обращаться по сети к записям в БД, размещенным на других компьютерах. В других случаях СУБД сама производит аутентификацию удаленного клиента и устанавливает сетевые соединения.

В общем случае режимы работы с БД можно классифицировать по следующим признакам:

- *количество одновременно выполняемых задач* — однопользовательский или многопользовательский;

- *правило обслуживания запросов* — последовательное или параллельное;
- *схема размещение данных* — централизованная или распределенная БД.

Следует отметить, что общая тенденция развития технологий обработки данных вполне соответствует этапам развития средств вычислительной техники и информационных технологий, и в первую очередь сетевых. В этом смысле следует выделить два класса: *системы распределенной обработки данных* и *системы распределенных баз данных*.

Системы распределенной обработки данных в основном отражают структуру и свойства многопользовательских операционных систем с базой данных, размещенной на большом центральном компьютере (мэйнфрейме). Еще до недавнего времени это был единственно возможный вариант вычислительной среды для реализации больших баз данных. Клиентские места в этом случае реализовывались в виде терминалов или мини-ЭВМ, обеспечивающих в основном ввод-вывод данных и не имеющих собственных вычислительных ресурсов для функционально-ориентированной обработки.

Развитие сетевых технологий в сочетании с широким распространением персональных ЭВМ и внедрением стандартов открытых систем привело к появлению систем баз данных, размещенных в сети разнотипных компьютеров. Такие *системы распределенных баз данных* обеспечивают обработку распределенных запросов, когда при обработке одного запроса используются ресурсы базы, размещенные на различных ЭВМ сети. Система распределенных баз данных состоит из узлов, каждый из которых является СУБД, а узлы взаимодействуют между собой так, что база данных любого узла будет доступна пользователю, как если бы она была локальной.

Соответственно, программы, обеспечивающие целевую (функциональную) обработку данных, могут быть организованы таким образом, чтобы обеспечить более эффективное использование совокупных вычислительных ресурсов за счет специализированного разделения функций обработки между центральным процессом СУБД и клиентскими функционально-ориентированными процедурами.

Для «типового» приложения обработки данных можно выделить следующие группы (уровни) функций:

- ввод и отображение данных: внешний (пользовательский) уровень реализации целевой функциональной обработки и представления (PL — Presentation Logic);

- функциональная обработка, реализующая алгоритм решения задач пользователя; соответствующие *«бизнес-правила»* реализуются обычно средствами высокоуровневого языка программирования или расширенного языка манипулирования данными типа ADABAS Natural или 4-GL (BL — Business Logic);
- манипулирование данными БД в рамках приложения, которое обычно реализуется средствами SQL (DBL — Database Logic). Кроме того, средствами SQL, помимо операций манипулирования данными (Data Management Logic — извлечения, изменения и т. д.) реализуются общие для БД функции (CDBL — Common DB Logic), например правила целостности, типовые представления, которые, по существу, являются общими «бизнес-правилами» на уровне данных;
- управление ресурсами БД, реализуемое специализированными средствами конкретной СУБД (RL — Resource Logic);
- управление процессами обработки: связывание и синхронизация процессов обработки данных разного уровня.

Рассматриваемые ниже архитектуры распределенной обработки в целом могут характеризоваться следующей диаграммой (рис. 11.1).

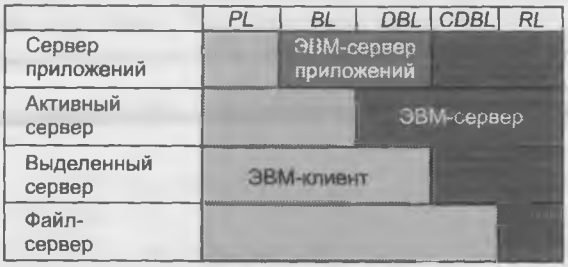


Рис. 11.1. Разделение функций в базовых архитектурах распределенной обработки

11.2. Архитектура распределенной обработки данных

Почти все модели организации взаимодействия пользователя с базой данных построены на основе модели «клиент—сервер». То есть предполагается, что каждое такое приложение отличается способом распределения функций ранее приведенных групп обработки данных между как минимум двумя частями:

- клиентской, которая отвечает за целевую обработку данных и организацию взаимодействия с пользователем;

- серверной, которая обеспечивает хранение данных, обрабатывает запросы и посылает результаты клиенту для специальной обработки.

В общем случае предполагается, что эти части приложения функционируют на отдельных компьютерах, т. е. к серверу БД с помощью сети подключены компьютеры пользователей (клиенты).

Сервер — это программа, реализующая функции собственно СУБД: определение данных, запись — чтение данных, поддержка схем внешнего, концептуального и внутреннего уровней, диспетчеризация и оптимизация выполнения запросов, защита данных.

Клиент — это различные программы, написанные как пользователями, так и поставщиками СУБД, внешние или «встроенные» по отношению к СУБД. Программа-клиент организована в виде приложения, работающего «поверх» СУБД и обращающегося для выполнения операций над данными к компонентам СУБД через интерфейс внешнего уровня[1].

Разделение процесса выполнения запроса на «клиентскую» и «серверную» компоненту позволяет:

- одновременно использовать общую базу данных различным прикладным (клиентским) программам;
- централизовать функции управления, такие как защита информации, обеспечение целостности данных, управление совместным использованием ресурсов;
- обеспечивать параллельную обработку запроса в случае распределенных БД;
- высвобождать ресурсы рабочих станций и сети;
- повышать эффективность управления данными за счет использования ЭВМ, специально разработанных для работы СУБД (серверы баз данных и машины баз данных).

11.2.1. Базовые архитектуры распределенной обработки

Учитывая, что одним из основных показателей эффективности сетевой обработки данных является время обслуживания запроса, рассмотрим различные модели архитектуры распределенной обработ-

[1] Инструментальные средства, в том числе и утилиты, не отнесены к серверной части очень условно. Являясь не менее важной составляющей, чем ядро СУБД, они выполняются самостоятельно как пользовательское приложение.

ки на примере, когда прикладная программа работы с базой данных, расположенной на сервере, загружена на рабочую станцию и пользователю необходимо получить все записи, удовлетворяющие некоторым поисковым условиям.

Архитектура «файл — сервер»

В архитектуре «файл — сервер», схема которой представлена на рис. 11.2, средства организации и управления базой данных (в том числе и СУБД) целиком располагаются на машине клиента, а база данных, представляющая собой обычно набор специализированных структурированных файлов, на машине-сервере. В этом случае серверная компонента представлена даже не средствами СУБД, а сетевыми составляющими операционной системы, обеспечивающими удаленный разделяемый доступ к файлам. Таким образом, «файл — сервер» представляет собой вырожденный случай клиент-серверной архитектуры.

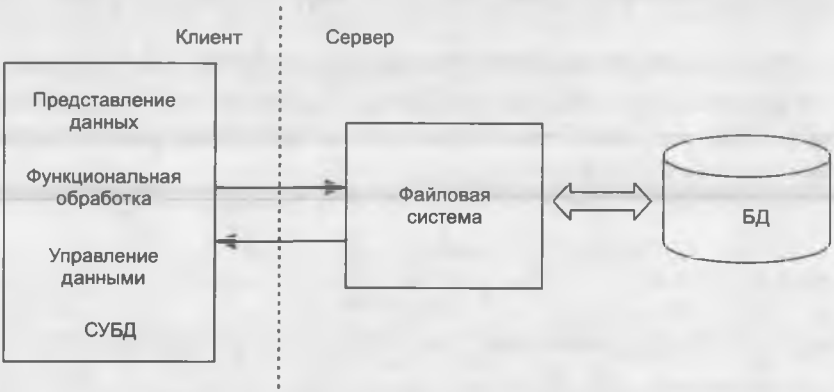


Рис. 11.2. Архитектура «файл — сервер»

Взаимодействие между клиентом и сервером происходит на уровне команд ввода-вывода файловой системы, которая возвращает запись или блок данных. Запрос к базе, сформулированный на языке манипулирования данными, преобразуется самой СУБД в последовательность команд ввода-вывода, которые обрабатываются операционной системой машины-сервера.

Достоинство — возможность обслуживания запросов нескольких клиентов.

Недостатки:

- высокая загрузка сети и машин-клиентов, так как обмен идет на уровне единиц информации файловой системы — физических записей, блоков или даже файлов, из которых на машине клиента будут выбраны и представлены необходимые для приложения элементы данных;
- низкий уровень защиты данных, так как доступ к файлам БД управляется общими средствами ОС-сервера;
- низкий уровень управления целостностью и непротиворечивостью информации, так как бизнес-правила функциональной обработки, сосредоточенные на клиентской части, могут быть противоречивыми и несинхронизированными.

В среде файлового сервера программа управления данными, которая выполняется на машине-клиенте, должна осуществить запрос каждой записи базы, после чего она может определить, удовлетворяет ли запись поисковым условиям, и лишь после этого передать запись для функциональной обработки. Очевидно, что для этой схемы характерно наибольшее суммарное время обработки информации.

Архитектура «выделенный сервер базы данных»

В архитектуре сервера базы данных, схема которой представлена на рис. 11.3, средства управления базой данных и база данных размещены на машине-сервере.

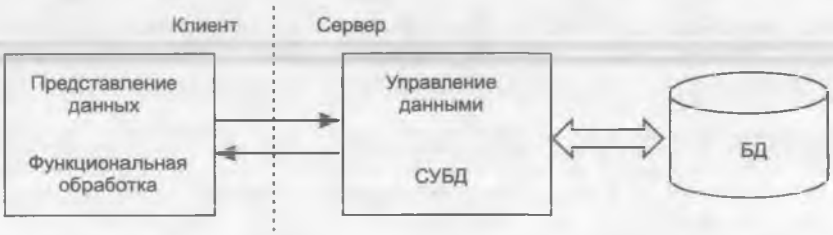


Рис. 11.3. Архитектура с выделенным сервером базы данных

Взаимодействие между клиентом и сервером происходит на уровне команд языка манипулирования данными СУБД (обычно SQL), которые обрабатываются СУБД на машине-сервере. Сервер базы данных осуществляет поиск записей и анализирует их. Записи, удовлетворяющие условиям, могут накапливаться на сервере, и после того, как запрос будет целиком обработан, пользователю на клиентскую

машину передаются все логические записи (запрашиваемые элементы данных), удовлетворяющие поисковым условиям.

Достоинства:

- возможность обслуживания запросов нескольких клиентов;
- снижение нагрузки на сеть и машины сервера и клиентов;
- защита данных осуществляется средствами СУБД, что позволяет блокировать не разрешенные пользователю действия;
- сервер реализует управление транзакциями и может блокировать попытки одновременного изменения одних и тех же записей.

Недостатки:

- бизнес-логика функциональной обработки и представление данных могут быть одинаковыми для нескольких клиентских приложений, и это увеличит совокупные потребности в ресурсах при исполнении вследствие повторения части кода программ и запросов;
- низкий уровень управления непротиворечивостью информации, так как бизнес-правила функциональной обработки, сосредоточенные на клиентской части, могут быть противоречивыми.

Данная технология позволяет снизить сетевой трафик и повысить общую эффективность обработки за счет оптимизации и буферизации ввода-вывода. Таким образом, сервер может осуществлять поиск и обрабатывать запросы даже быстрее, чем если бы они обрабатывались на рабочей станции.

Архитектура «активный сервер баз данных»

Для того чтобы устранить недостатки, свойственные архитектуре сервера базы данных, необходимо, чтобы непротиворечивость бизнес-логики и изменения базы данных контролировались на стороне сервера. Причем некоторые заранее специфицированные состояния могли бы изменять последовательность взаимодействия приложения с базой данных.

Для этого функции бизнес-логики разделяются между клиентской и серверной частями. Общие или критически значимые функции оформляются в виде *хранимых процедур*, включаемых в состав базы данных. Кроме этого вводится механизм отслеживания событий БД — *триггеров*, также включаемых в состав базы. При возникновении соответствующего события (обычно изменения данных)

СУБД вызывает для выполнения хранимую процедуру, связанную с триггером, что позволяет эффективно контролировать изменение базы данных.

Хранимые процедуры и триггеры могут быть использованы любыми клиентскими приложениями, работающими с базой данных. Это снижает дублирование программных кодов и исключает необходимость компиляции каждого запроса (рис. 11.4).

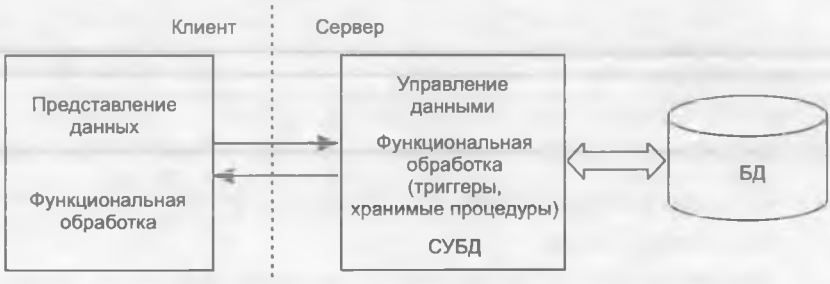


Рис. 11.4. Архитектура «активный сервер баз данных»

Недостатком такой архитектуры становится существенно возрастающая загрузка сервера за счет необходимости отслеживания событий и выполнения части бизнес-правил.

Такую архитектуру организации взаимодействия (а также рассматриваемый далее сервер приложений) иногда называют *моделью с «тонким клиентом»*, в отличие от предыдущих архитектур, называемых *моделью с «толстым клиентом»*, где на стороне клиента выполняется большинство функций.

Архитектура «сервер приложений»

Рассмотренные выше архитектуры являются *двухзвенными*: здесь все функции доступа и обработки распределены между программой клиента и сервером БД.

Дальнейшее снижение уровня требований к ресурсам клиента достигается за счет введения промежуточного звена — *сервера приложений*, на который переносится значительная часть программных компонентов управления данными и бóльшая часть бизнес-логики. При этом серверы баз данных обеспечивают исключительно функции СУБД по ведению и обслуживанию базы данных. Схема трехзвенной архитектуры сервера приложений приведена на рис. 11.5.

Рис. 11.5. Архитектура сервера приложений

К другим (организационно-технологическим) достоинствам трехзвенной архитектуры можно отнести:

- централизованное ведение бизнес-логики и в случае внесения изменений отсутствие необходимости их тиражирования в клиентских приложениях;
- отсутствие необходимости устанавливать на клиентских машинах компонент программного обеспечения управления доступом к данным;
- возможность отложенного обновления БД в случае изменения данных, запрошенных с сервера, в автономном режиме. Данные будут обновлены в базе после следующего соединения клиентской программы с сервером приложений.

11.2.2. Архитектура сервера баз данных

Повышение эффективности и оперативности обслуживания большого числа клиентских запросов, помимо простого увеличения ресурсов и вычислительной мощности серверной машины, может быть достигнуто двумя путями:

- снижением суммарного расхода памяти и вычислительных ресурсов за счет буферизации (кэширования) и совместного использования наиболее часто запрашиваемых данных и процедур (разделяемые ресурсы);
- распараллеливанием процесса обработки запроса — использованием разных процессоров для параллельной обработки изолированных подзапросов и/или для одновременного обращения к частям базы данных, размещенным на отдельных физических носителях.

Рассмотрим архитектуры, реализующие следующие модели совместной обработки клиентских запросов.

Архитектура «один к одному»

В этом случае (рис. 11.6) для обслуживания каждого запроса запускается отдельный серверный процесс.

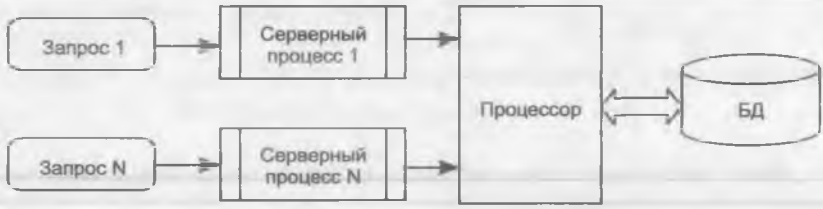


Рис. 11.6. Архитектура сервера «один к одному»

Таким образом, даже если от клиентов поступят совершенно одинаковые запросы, для обработки каждого из них будут запущены отдельные процессы, каждый из которых будет выполнять одинаковые для всех запросов действия и использовать одни и те же ресурсы.

Многопотоковая односерверная архитектура

Обработку всех клиентских запросов выполняет один серверный процесс (использующий один процессор), взаимодействующий со всеми клиентами и монопольно управляющий ресурсами (рис. 11.7). При этом для отдельного клиентского процесса создается поток (thread), в рамках которого локализуется обработка запроса.

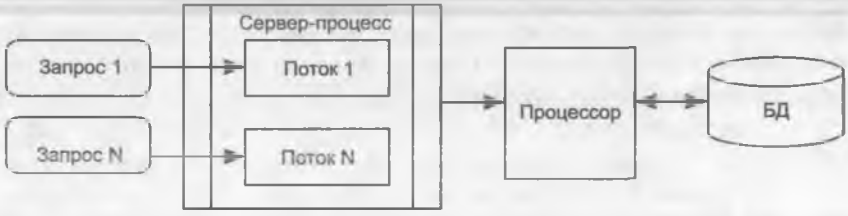


Рис. 11.7. Многопотоковая односерверная архитектура

11.2.2.3. Мультисерверная архитектура

В том случае, когда для работы СУБД используются многопроцессорные платформы, обслуживание запросов может быть физически распределено для параллельной обработки между процессорами (рис. 11.8). Такое решение требует введения дополнительного звена, в

задачи которого входит диспетчеризация запросов для обеспечения сбалансированной загрузки процессоров.

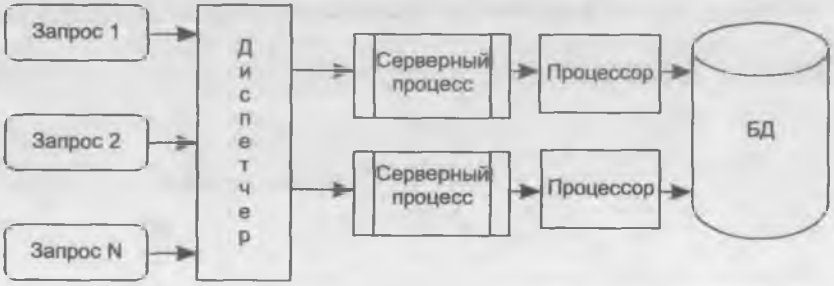


Рис. 11.8. Многопотоковая односерверная архитектура

Если серверный процесс реализуется как многопоточное приложение, говорят, что СУБД имеет *мультисерверную многопотоковую архитектуру*.

Следует отметить, что характер распределения запросов в значительной степени зависит от того, поддерживает ли операционная система потоковую обработку, а также от возможностей средств управления приоритетами задач.

Серверные архитектуры с параллельной обработкой запроса

Для повышения оперативности за счет распараллеливания процесса обработки отдельного клиентского запроса в мультисерверной архитектуре можно использовать следующие подходы.

1. Размещение хранимых данных БД на нескольких физических носителях (сегментирование базы). Для обработки запроса в этом случае запускаются несколько серверных процессов (использующих обычно отдельные процессоры), каждый из которых независимо от других выполняет одинаковую последовательность действий, определяемую существом запроса, но с данными, принадлежащими разным сегментам базы. Полученные таким образом результаты объединяются и передаются клиенту. Такой тип распараллеливания называют *моделью горизонтального параллелизма*.

2. Запрос обрабатывается по конвейерной технологии. Для этого запрос разбивается на взаимосвязанные по результатам подзапросы, каждый из которых может быть обслужен отдельным серверным процессом независимо от обработки других подзапросов. Получаемые

результаты объединяются согласно схеме декомпозиции запроса и передаются клиенту. Такой тип распараллеливания называют *моделью вертикального параллелизма*.

Примерная схема обработки клиентского запроса, построенная с использованием обеих моделей параллелизма (гибридная модель), приведена на рис. 11.9.

Рис. 11.9. Архитектура сервера обработки запроса при гибридном параллелизме

Использование моделей параллельной обработки позволяет существенно сократить общее время обслуживания запроса, что особенно важно в случае работы с большими базами данных и аналитической обработки (OLAP-приложений).

11.3. Технологии и средства доступа к удаленным БД

11.3.1. Программное обеспечение распределенных приложений

Распределенные корпоративные приложения все более усложняются, интегрируя унаследованные приложения, разрабатываемые и вновь приобретаемые готовые программные средства. Кроме того, разные подсистемы решают разные бизнес-задачи, однако одна из главных целей создания корпоративной системы — получить «единый образ» общего состояния системы, что обеспечит пользователям доступ к нужным операциям и ресурсам.

Основа такой инфраструктуры — так называемое *промежуточное программное обеспечение*, позволяющее, не вникая в тонкости сетевых реализаций, создавать и эксплуатировать взаимодействующие приложения с разными требованиями к межмодульным коммуникациям.

Промежуточное ПО эволюционировало вместе с архитектурой «клиент — сервер». Ранние, но достаточно эффективные как с точки зрения разработки, так и в эксплуатации частные решения предназначались для упрощения доступа к базам данных в двухзвенной модели, где «толстый» клиент реализует всю логику обработки информации, предоставляемой сервером базы данных. Такие системы вполне удовлетворяли потребностям небольших корпоративных подразделений с ограниченным числом пользователей и невысокой интенсивностью обмена.

Однако по мере того, как клиент-серверная архитектура стала проникать в сферу высококритичных корпоративных приложений, обслуживающих уже не десятки, а сотни пользователей и работающих со значительными массивами данных, стали очевидны недостатки двухзвенного подхода. Этот способ реализации клиент-серверной схемы доступа ограничивал возможности масштабирования, поскольку рост числа обращений к одной базе данных непомерно увеличивал нагрузку на сервер и делал доступ к данным узким местом в общей производительности системы. Кроме того, всякая модификация логики приложения требовала внесения изменений во все экземпляры клиентских приложений.

Чтобы избежать таких проблем, для разработки корпоративных приложений используют трехзвенную модель, которая переносит логику приложения на отдельный уровень сервера приложений. В результате клиентская часть приложения становится «тоньше» и в основном отвечает за предоставление удобного пользовательского интерфейса. Как правило, сервер баз данных также освобождается от необходимости поддерживать бизнес-логику, которая в двухзвенной модели реализуется с помощью специальных расширений СУБД, например хранимых процедур. Перенос основных операций приложения на отдельный уровень позволяет с максимальной эффективностью распределить нагрузку на аппаратные средства (трехзвенная модель на самом деле может быть многозвенной с разделением нагрузки на несколько серверов приложений) и обеспечивает безболезненное наращивание как функциональности приложения, так и числа обслуживаемых пользователей.

Развитие этого среднего звена клиент-серверной модели идет в сторону усложнения. Ограничиваясь вначале построением более высокого уровня абстракции для взаимодействия приложения с ресурсами данных, разработчик приложения получал возможность использовать общие API (Application Program Interface), которые скрывали различия специфических интерфейсов коммуникационных протоколов более низкого уровня, например TCP/IP, Sockets или DECNet. Однако теперь этого уже явно недостаточно для построения сложных распределенных приложений. Современные решения не только обеспечивают межпрограммное взаимодействие, но и являются платформой для реализации сервера приложений, обеспечивая обширный набор необходимых служб: управления транзакциями, именования, защиты и т. д.

Вычислительная среда распределенных приложений может включать в себя различные операционные системы, аппаратные платформы, коммуникационные протоколы и разнообразные средства разработки. Соответственно формат представления данных в различных узлах будет различаться.

Таким образом, в распределенной неоднородной среде программное обеспечение промежуточного уровня играет роль «информационной шины», надстроенной над сетевым уровнем и обеспечивающей

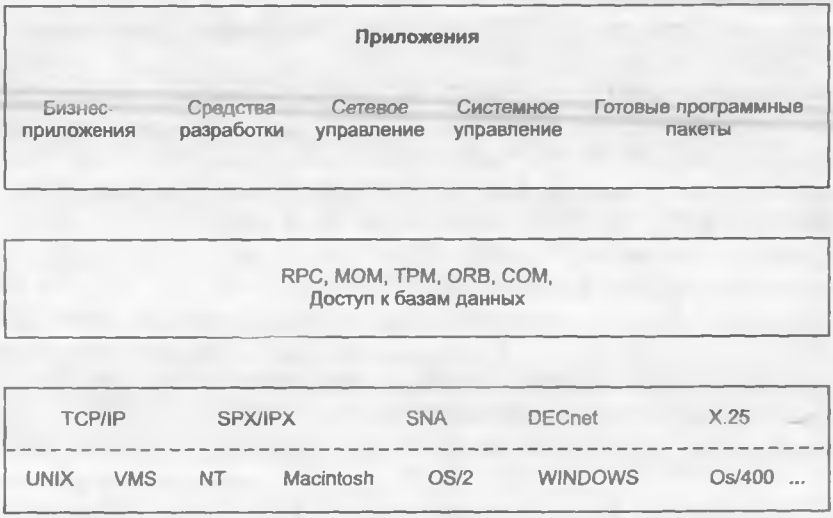


Рис. 11.10. Структура компонентов поддержки удаленного доступа

доступ приложения к разнородным ресурсам, а также независимую от платформ взаимосвязь различных прикладных компонентов, изолирующую логику приложений от уровня сетевого взаимодействия и ОС (рис. 11.10).

ПО промежуточного уровня можно разделить на две категории.

1. ПО доступа к базам данных (например, ODBC-интерфейсы и SQL-шлюзы).

2. ПО межмодульного взаимодействия — системы, реализующие вызов удаленных процедур (RPC — Remote Procedure Call); мониторы обработки транзакций (TP-мониторы); средства интеграции распределенных объектов.

При этом следует отметить, что различия прикладных задач не позволяют построить универсальное ПО, реализовав в одном продукте все необходимые возможности.

11.3.2. Доступ к базам данных в двухзвенных моделях «клиент—сервер»

В простых двухзвенных моделях «клиент—сервер», где несколько баз данных обслуживают ограниченное число пользователей настольных ПК, в роли встроенного ПО доступа к данным могут выступать обычные ODBC-драйверы.

Необходимость в более сложных решениях возникает в больших разнородных многозвенных системах, где множество приложений в параллельном режиме осуществляют доступ к разнообразным источникам данных, включая разнотипные СУБД и хранилища данных. В таких системах между клиентами и серверами баз данных размещается промежуточное звено — SQL-шлюз, который представляет собой набор общих API, позволяющих разработчику строить унифицированные запросы к разнородным данным (в формате SQL или с помощью ODBC-интерфейса). SQL-шлюз выполняет синтаксический разбор такого запроса, анализирует и оптимизирует его и в конце концов выполняет преобразование в SQL-диалект нужной СУБД. ПО этого типа реализует синхронный механизм связи, когда выполнение приложения, сделавшего запрос, блокируется до момента получения данных.

Примером такого приложения может быть система анализа статистических данных о деятельности компаний, которая отбирает соответствующую информацию из расположенных в различных регио-

нах баз данных с разными СУБД. Подобные решения достаточно просты, не требуют сложных механизмов управления транзакциями и способны обеспечить постепенную миграцию важных приложений с унаследованных платформ в архитектуру «клиент—сервер».

Создается такое приложение обычно с использованием средств языков высокого уровня (например, C++, Pascal, Visual Basic), позволяющих реализовать эффективную целевую обработку данных и дружественный пользовательский интерфейс. В исходный текст программы включаются SQL-выражения, специфицирующие условия выборки или изменения данных в базе. Во время исполнения приложения эти выражения передаются серверу, который, собственно, и манипулирует данными. Данные, полученные в результате выполнения сервером SQL-запросов, возвращаются прикладной программе и размещаются в заранее определенных структурах для дальнейшей обработки, в том числе корректировки записей.

Рассмотрим различные способы организации доступа прикладной программы к серверу базы данных в двухзвенной архитектуре.

Использование библиотек доступа и встраиваемого SQL

Каждая СУБД помимо интерактивной SQL-утилиты обязательно имеет библиотеку процедур доступа и набор драйверов СУБД для различных операционных систем. Схема взаимодействия клиентского приложения с сервером базы данных в этом случае представлена на рис. 11.11.

Библиотека доступа содержит набор функций, позволяющих клиентскому приложению соединяться с базой данных, передавать запросы серверу и получать данные — результаты обработки запроса. Типичный набор функций такой библиотеки включает:

- соединение с базой данных;
- запрос на добавление данных;
- запрос на извлечение данных;
- запрос на изменение данных;
- закрытие соединения с базой данных.

Обычно в библиотеке присутствуют также функции, позволяющие определить характеристики структуры набора результата (число, порядок и имена столбцов, число строк, номер текущей строки), передвигаться по этой структуре не только вперед, но и назад и т. д.

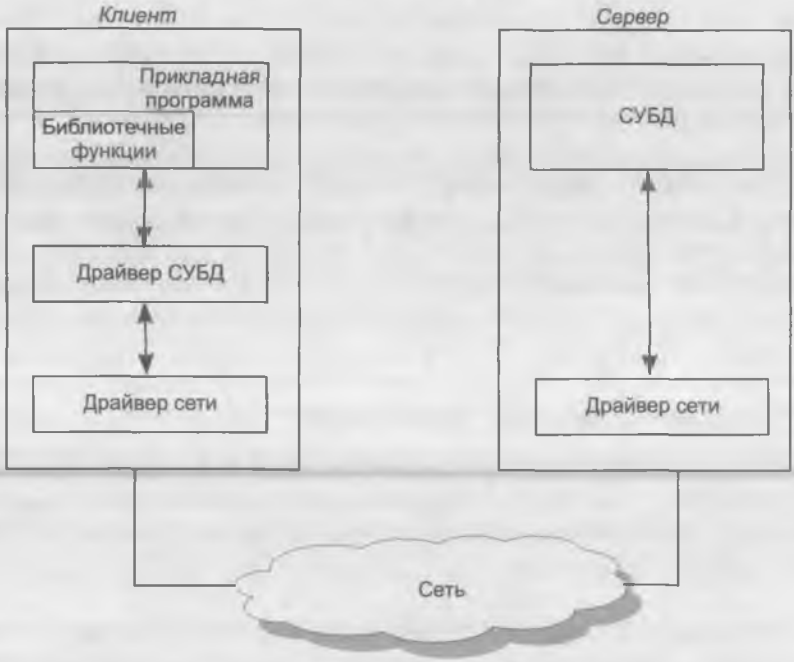


Рис. 11.11. Схема взаимодействия с использованием библиотек процедур доступа

Библиотечные вызовы преобразуются драйвером базы данных в сетевые вызовы и передаются сетевым программным обеспечением на сервер. На сервере происходит обратный процесс преобразования сетевых пакетов в SQL-запросы, которые обрабатываются СУБД. Результаты обработки передаются клиенту.

Такой способ создания приложений достаточно гибок и позволяет реализовать практически любое приложение, однако имеет и недостатки:

- разработка клиентской программы возможна только для той операционной системы и на том языке программирования, в которых поддерживается библиотека;
- драйвер базы данных определяет допустимые типы сетевых интерфейсов;
- библиотечные функции обычно не унифицированы.

Некоторой модификацией данного способа является использование «встроенного» языка SQL. В этом случае текст программы на языке третьего поколения вместо вызовов функций библиотеки

включает непосредственно предложения SQL, которые предваряются выражением «EXEC SQL». Перед компиляцией в машинный код такая программа обрабатывается препроцессором, который транслирует смесь операторов «собственного» языка СУБД и SQL-предложений в промежуточный «чистый» исходный код, а затем коды SQL замещаются вызовами соответствующих процедур из библиотек, поддерживающих конкретную СУБД. Такой подход позволяет несколько снизить степень привязанности к СУБД; например, при переключении прикладной программы на работу с другим сервером базы данных достаточно указать новый сервер и заново перекомпилировать программу.

Программный интерфейс уровня вызовов

Стандарт SQL2 определил интерфейс уровня вызова (CLI — Call Level Interface), в котором стандартизирован общий набор рабочих процедур, обеспечивающий совместимость со всеми основными типами серверов баз данных.

Технологическая основа CLI — размещаемая на компьютере клиента специальная библиотека, в которой хранятся вызовы процедур и сетевых компонентов для организации связи с сервером. Это программное обеспечение поставляется обычно в составе среды разработки и поддерживает разнообразные сетевые протоколы.

Использование программных вызовов позволяет свести к минимуму операции на компьютере-клиенте. В общем случае клиент формирует оператор языка SQL в виде строки и пересылает ее на сервер посредством процедуры исполнения (execute). Когда же сервер в качестве ответа возвращает несколько строк данных, клиент считывает результат последовательным вызовом процедуры выборки данных. Далее данные из столбцов полученной таблицы могут быть связаны с соответствующими переменными приложения. Вызов специальной процедуры позволяет клиенту определить число полученных строк, столбцов и типы данных в каждом столбце.

Открытый интерфейс доступа к базам данных

Спецификация открытого интерфейса баз данных (ODBC — Open Database Connectivity) предназначена для унификации доступа к данным, размещенным на удаленных серверах. ODBC опирается на спецификации CLI.

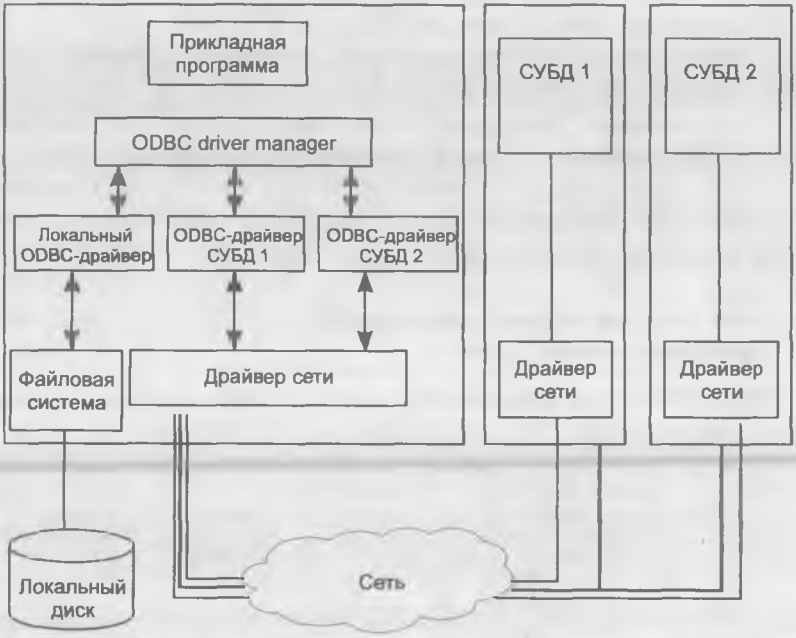


Рис. 11.12. Структурная схема доступа к данным с использованием ODBC

ODBC представляет собой программный слой, унифицирующий интерфейс взаимодействия приложений с базами данных. За реализацию особенностей доступа к каждой отдельной СУБД отвечает соответствующий специальный ODBC-драйвер. Пользовательское приложение этих особенностей не видит, так как взаимодействует с универсальным программным слоем более высокого уровня. Таким образом, приложение становится в значительной степени независимым от СУБД. Вместо создания в каждом отдельном случае СУБД-приложения с обращениями через «родной», но быстро устаревающий интерфейс можно использовать один общий стандартизированный программный интерфейс.

В архитектуре ODBC используется один ODBC Driver Manager и несколько ODBC-драйверов, обеспечивающих доступ к конкретным СУБД. Driver Manager связывает приложение и интерфейсные объекты, которые выполняют обработку SQL-запросов к конкретной СУБД.

Такой подход является достаточно универсальным, стандартизируемым, что и позволяет использовать ODBC-механизмы для работы практически с любой системой.

Однако этот способ также не лишен недостатков:

- увеличивается время обработки запросов (как следствие введения дополнительного программного слоя);
- необходимы предварительная инсталляция и настройка ODBC-драйвера (указание драйвера СУБД, сетевого пути к серверу, базы данных и т. д.) на каждом рабочем месте. Параметры этой настройки являются статическими, т. е. приложение их изменить самостоятельно не может.

Мобильный интерфейс к базам данных на платформе Java

JDBC (Java Data Base Connectivity) — это интерфейс прикладного программирования (API) для выполнения SQL-запросов к базам данных из программ, написанных на платформенно-независимом

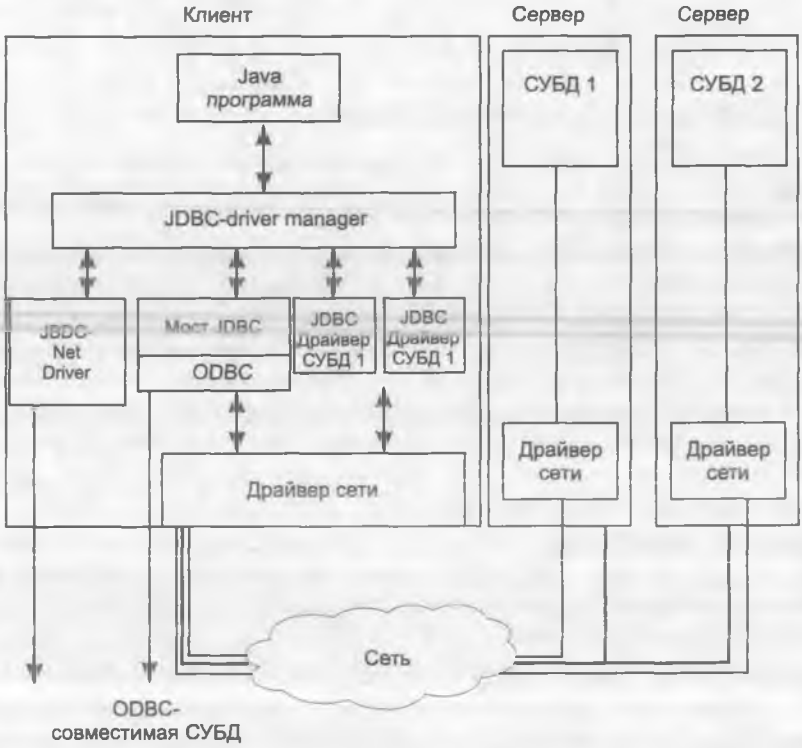


Рис. 11.13. Структурная схема доступа к данным с использованием JDBC

языке Java, позволяющем создавать как самостоятельные приложения (standalone application), так и апплеты, встраиваемые в Web-страницы.

JDBC во многом подобен ODBC, он также построен на основе спецификации CLI, однако имеет ряд следующих отличий:

- приложение загружает JDBC-драйвер динамически, следовательно, администрирование клиентов упрощается, более того, появляется возможность переключаться на работу с другой СУБД без перенастройки клиентского рабочего места;
- JDBC, как и Java в целом, не привязан к конкретной аппаратной платформе, следовательно, проблемы с переносимостью приложений практически снимаются;
- использование Java-приложений и связанной с ними идеологии «тонких клиентов» позволяет снизить требования к оборудованию клиентских рабочих мест.

Обобщенная структурная схема доступа к данным с использованием JDBC приведена на рис. 11.13.

Прикладные интерфейсы OLE DB и ADO

Встраивание и связывание объектов в базах данных — OLE DB (Object Linking and Embedding Data Base), как и ODBC, — прикладной интерфейс доступа к данным с использованием SQL.

OLE DB специфицирует взаимодействие, обеспечивая единый интерфейс доступа к данным через провайдеров — поставщиков данных не только из реляционных БД. В отличие от ODBC, OLE DB предоставляет общее решение обеспечения COM-приложениям доступа к информации независимо от типа источника данных.

OLE DB включает два базовых компонента: *провайдер данных* и *потребитель данных*. Потребитель (клиент) — это приложение или COM-компонент, обращающийся посредством API-вызовов к OLE DB. Провайдер (сервер) — это приложение, отвечающее на вызовы OLE DB и возвращающее запрашиваемый объект (обычно это данные в табличном виде).

ADO (Active Data Object) — это универсальный интерфейс высокого уровня к OLE DB. Модель объекта ADO не содержит таблиц, среды или машины БД. Здесь основными объектами являются следующие: объект *Соединение*, создающий связь с провайдером данных; объект *Набор данных* и объект *Команда* — выполнение процедуры или SQL-строки.

В общем случае ADO можно рассматривать как язык программирования операций с БД, позволяющий выбирать, модифицировать и удалять записи. И поскольку он опирается на универсальный OLE DB, то может использоваться практически в любых приложения Microsoft.

Взаимосвязь механизмов доступа к данным

Один из способов организации доступа к данным заключается в непосредственном использовании API. Однако это означает полную зависимость создаваемого приложения от используемой СУБД. В этом случае переход к другой системе (например, от настольной системы к системе типа клиент/сервер) влечет за собой переписывание большей части программного кода клиентского приложения.

Таким образом, следующим этапом в обеспечении доступа клиентского приложения к данным является создание универсального механизма доступа к БД, обеспечивающего для клиентского приложения стандартный набор функций, классов или сервисов (служб), необходимых для работы с различными системами управления базами данных. Эти стандартные функции (классы или сервисы) должны размещаться в библиотеках, именуемых *драйверами* или *провайдерами баз данных* (data base drivers (providers)). Каждая такая библиотека реализует набор стандартных функций, классов или сервисов, используя обращения API к конкретной СУБД.

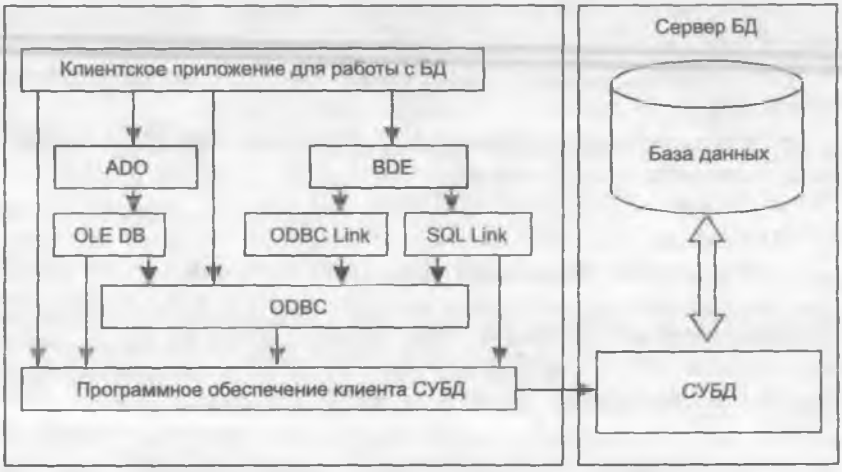


Рис. 11.14. Взаимосвязь механизмов доступа к данным

Наиболее популярными механизмами доступа к данным (Universal Data Access, UDA) в настоящий момент являются:

- ODBC;
- OLE DB;
- ADO;
- BDE.

Первые три являются фактически промышленными стандартами. Последний долгое время был единственным механизмом доступа к данным, реализованным в инструментальных средствах разработки компании Borland (например, Delphi, C++Builder), однако в последних версиях инструментальных средств не развивается.

На рис. 11.14 схематически представлены различные механизмы доступа к данным, включая непосредственные вызовы клиентской частью API системы управления базой данных.

11.4. Корпоративные серверы приложений

Появление серверов приложений как отдельных готовых решений связано и с бурным вторжением Web-технологий в сферу корпоративных высококритичных систем. Однако возможности протокола HTTP ограничены функциями связи без каких-либо средств сохранения информации о состоянии, поэтому он не подходит для поддержки мощных корпоративных систем.

На рис. 11.15 приведен «идеальный» состав сервера приложений с максимальным набором необходимых служб и средств связи с клиентскими системами и информационными ресурсами.

Сегодня прикладные разработки базируются на одной из двухкомпонентных моделей — MTS/DCOM и CORBA, способных интегрировать объекты на удаленных платформах.

Обе модели распространяют принципы вызова удаленных процедур на объектные распределенные приложения и обеспечивают прозрачность реализации и физического размещения серверного объекта для клиентской части приложения; поддерживают возможность взаимодействия объектов, созданных на различных объектно ориентированных языках, и скрывают от приложения детали сетевого взаимодействия.

В DCOM взаимодействие удаленных объектов, представленное на рис. 11.16, базируется на спецификации DCE RPC, а CORBA использует брокер объектных запросов (ORB), синхронный механизм которого во многом схож с RPC.

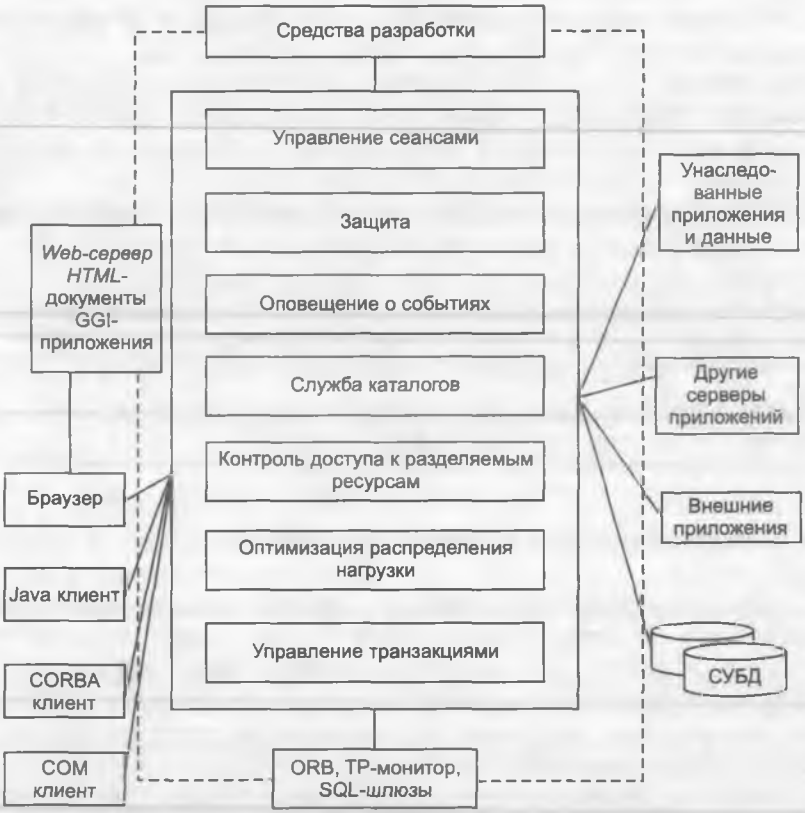


Рис. 11.15. Обобщенная структура сервера приложений

Рис. 11.16. DCOM-технология взаимодействия «клиент — сервер»

В DCOM-технологии взаимодействие между клиентом и сервером осуществляется через двух посредников. Клиент помещает параметры вызова в стек и обращается к методу интерфейса объекта. Это обращение перехватывает посредник Proxy, упаковывает параметры вызова в COM-пакет и адресует его в Stub, который в свою очередь распаковывает параметры в стек и инициирует выполнение метода объекта в пространстве сервера.

CORBA-технология также использует интерфейс объекта, но в этом случае схема взаимодействия объектов (рис. 11.17) включает промежуточное звено (*Smart agent*), реализующее доступ к удаленным объектам. Smart agent, установленный на машинах сетевого окружения (сервере локальной сети или Internet-узле), моделирует сетевой каталог известных ему серверов объектов. При создании сервера происходит автоматическая регистрация его объектов в каталоге одного или нескольких Smart agent.

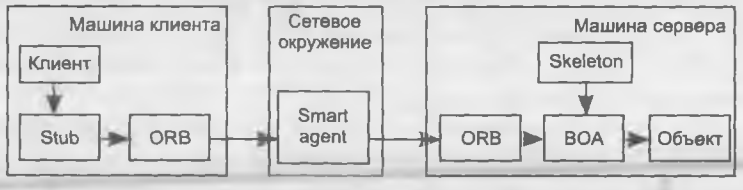


Рис. 11.17. CORBA-технология взаимодействия «клиент—сервер»

Связи между брокерами осуществляются в соответствии с требованиями специального протокола General Inter ORB Protocol, определяющего низкоуровневое представление данных и множество форматов сообщений.

На машине клиента создаются два объекта-посредника: Stab и ORB (Object Required Broker — брокер вызываемого объекта). Так же как и в DCOM-технологии, Stub передает перехваченный вызов брокеру, который посылает широковещательное сообщение в сеть. Smart agent, получив сообщение, отыскивает сетевой адрес сервера и передает запрос брокеру, размещенному на машине сервера. Вызов требуемого объекта производится через специальный базовый объектный адаптер (BOA). При этом данные в стек пространства вызываемого объекта помещает особый объект сервера (Skeleton), который вызывается адаптером.

Кроме того, CORBA, помимо механизма взаимодействия с помощью ORB, включает в себя ряд общих служб CORBA Services (службы каталогов, защиты, оповещения о событиях, поддержки транзакций и ряд других), а также реализаций объектов для разных прикладных областей.

Ключевым компонентом архитектуры CORBA является язык описания интерфейсов IDL, на уровне которого поддерживаются «контрактные» отношения между клиентом и сервером и обеспечивается независимость от конкретного объектно ориентированного язы-

ка. CORBA IDL поддерживает основные понятия объектно ориентированной парадигмы (инкапсуляцию, полиморфизм и наследование).

В модели DCOM также может использоваться разработанный Microsoft язык IDL, который, однако, играет вспомогательную роль и используется в основном для удобства описания объектов. Реальная интеграция объектов в DCOM происходит не на уровне абстрактных интерфейсов, а на уровне бинарных кодов, и это одно из основных различий этих двух объектных моделей.

И DCOM, и CORBA, в отличие от процедурного RPC, дают возможность динамического связывания удаленных объектов: клиент может обратиться к серверу-объекту во время выполнения, не имея информации об этом объекте на этапе компиляции. В CORBA для этого существует специальный интерфейс динамического вызова DII, а COM использует механизм OLE-Automation. Информацию о доступных объектах сервера на этапе выполнения клиентская часть программы получает из специального хранилища метаданных об объектах — репозитария интерфейсов Interface Repositary в случае CORBA или библиотеки типов (Type Library) в модели DCOM. Эта возможность очень важна для больших распределенных приложений, поскольку позволяет менять и расширять функциональность серверов, не внося существенных изменений в код клиентских компонентов программы. Пример — банковское приложение, основная бизнес-логика которого поддерживается сервером в центральном офисе, а клиентские системы разбросаны по филиалам в разных городах.

11.5. Доступ к данным с помощью ADO.NET

ADO.NET является преемником Microsoft ActiveX Data Objects (ADO). Это W3C-стандартизированная модель программирования для создания распределенных прикладных программ, нацеленных на совместное использование данных.

ADO.NET является программным интерфейсом (API) для прикладного программного обеспечения, позволяющим обращаться к данным и другой информации. ADO.NET поддерживает такие современные требования, как создание клиентского интерфейса к базам данных на фронтальном уровне и на уровне промежуточного слоя объектов клиентских приложений, инструментальных средств, языков программирования или Internet-браузера.

ADO.NET, подобно ADO, обеспечивает интерфейс доступа к OLE DB-совместимым источникам данных. Прикладные программы, позволяющие пользователям совместно использовать данные, могут использовать ADO.NET для подключения к источникам данных, а также для поиска и модификации этих данных. Прикладные программы также могут использовать OLE DB для управления данными, хранящимися в форматах, отличных от форматов БД.

В решениях, требующих автономного или удаленного доступа к данным, ADO.NET использует XML для обмена данными между программами или с Web-страницами. Любой компонент, который обслуживает XML, также может использовать и компоненты ADO.NET. Если передача пакетов компонентом ADO.NET подразумевает поставку набора данных в файле XML, то компонентом, способным обеспечить его получение, может быть только компонент ADO.NET. Передача данных в XML-формате дает возможность легко отделить обработку данных от компонентов пользовательского интерфейса.

Для распределенных приложений использование наборов данных XML в ADO.NET обеспечивает бо́льшую эффективность, чем использование COM для офлайнового обслуживания данных в ADO. Поскольку передача наборов данных происходит через файлы XML, описанные в достаточно простом стандартном языке и являющиеся обычными текстовыми файлами, компоненты ADO.NET не имеют архитектурных ограничений, свойственных COM. Фактически любые два компонента могут совместно использовать наборы XML-данных при условии, что они оба используют ту же самую XML-схему форматирования.

ADO.NET обладает хорошей масштабируемостью, что удобно для совместно использующих данные Web-приложений. Кроме того, ADO.NET не использует длительные блокировки баз данных и активные подключения, которые на долгое время монополизируют ресурсы сервера, являющиеся, как правило, весьма ограниченными. Это позволяет увеличивать число пользователей без значительного увеличения загрузки ресурсов системы.

Контрольные задания

1. Сформулируйте основные требования к системам управления распределенными базами данных.
2. Перечислите основные условия и предпосылки появления систем управления распределенными базами данных.

3. Перечислите основные различия системы распределенной обработки данных и системы распределенных баз данных.
4. Обоснуйте целесообразность разделения «клиентских» и «серверных» функций.
5. Проведите сравнительный анализ распределения функций для различных базовых архитектур.
6. Определите основные принципы и примерные структурные схемы сервера распределенной обработки.
7. Перечислите основные решения распределенной обработки на основе межмодульного взаимодействия.

Глава 12
ТРАНЗАКЦИИ И ЦЕЛОСТНОСТЬ БД

Применение СУБД для работы с интегрированными БД выявило особую важность проблемы *целостности* БД. Под целостностью БД понимают правильность и непротиворечивость ее содержимого. Нарушение целостности может быть вызвано, например, ошибками и сбоями, так как в этом случае система не в состоянии обеспечить нормальную обработку или выдачу правильных данных.

Рассмотрим два аспекта целостности — на уровне отдельных объектов и операций и на уровне базы данных в целом.

Первый аспект целостности обеспечивается на уровне структур данных и отдельных операторов языковых средств СУБД (вспомним ограничения целостности для столбцов и таблиц в языке SQL). При нарушениях такой целостности (например, ввод значения больше 10 в столбец *Семестр* таблицы «Учебный_план» БД «Сессия») соответствующий оператор отвергается.

Некоторые ограничения целостности не нужно выражать в явном виде, поскольку они встроены в структуры данных. Например, в СУБД, поддерживающей структуры, составленные из записей, каждый экземпляр записи в БД должен отображать спецификацию типа записи. Это означает, что все поля, специфицированные в описании типа, должны быть представлены в каждом экземпляре записи, а значение, заносимое в отдельное поле, должно иметь соответствующий описанию тип данных.

Часто же база данных может иметь такие ограничения целостности, которые требуют обязательного выполнения не одной, а нескольких операций. Для иллюстрации примеров этой главы расширим функциональные возможности учебной БД «Сессия», добавив в таблицу «Кадровый_состав» столбец *Нагрузка* для решения дополнительной задачи — расчета общей годовой нагрузки преподавателей (в часах учебной работы). Тогда любая операция по внесению изменений или по добавлению данных в столбец *ID_Преподаватель* таблицы

«Учебный_план» должна сопровождаться соответствующими изменениями данных в столбце *Нагрузка*. Если после внесения изменений в столбец *ID_Преподаватель* произойдет сбой, то БД окажется в нецелостном состоянии.

Для обеспечения целостности в случае ограничений на базу данных, а не на какие-либо отдельные операции служит аппарат транзакций.

Транзакция — неделимая с точки зрения воздействия на БД последовательность операторов манипулирования данными (чтения, удаления, вставки, модификации), такая, что:

1) либо результаты всех операторов, входящих в транзакцию, отображаются в БД;

2) либо воздействие всех этих операторов полностью отсутствует.

При этом для поддержания ограничений целостности на уровне БД допускается их нарушение внутри транзакции так, чтобы к моменту завершения транзакции условия целостности были соблюдены.

Для обеспечения контроля целостности каждая транзакция должна начинаться при целостном состоянии БД и должна сохранить это состояние целостным после своего завершения. Если операторы, объединенные в транзакцию, выполняются, то происходит нормальное завершение транзакции и БД переходит в обновленное (целостное) состояние (ситуация COMMIT на рис. 12.1). Если же происходит сбой при выполнении транзакции, то происходит так называемый откат к исходному состоянию БД (ситуация ROLLBACK на рис. 12.1).

Рис. 12.1. Выполнение и откат транзакции

12.1. Модели транзакций

Рассмотрим две модели транзакций, используемые в большинстве коммерческих СУБД: модель автоматического выполнения транзакций и модель управляемого выполнения транзакций, обе основанные на инструкциях языка SQL — COMMIT и ROLLBACK.

12.1.1. Автоматическое выполнение транзакций

В стандарте ANSI/ISO зафиксировано, что транзакция автоматически начинается с выполнения пользователем или программой первой инструкции SQL. Далее происходит последовательное выполнение инструкций до тех пор, пока транзакция не завершается одним из двух способов (рис. 12.2):

- инструкцией COMMIT, которая выполняет завершение транзакции: изменения, внесенные в БД, становятся постоянными, а новая транзакция начинается сразу после инструкции COMMIT;
- инструкцией ROLLBACK, которая отменяет выполнение текущей транзакции и возвращает БД к состоянию начала транзакции, новая транзакция начинается сразу после инструкции ROLLBACK.

Такая модель создана на основе модели, принятой в СУБД DB2.

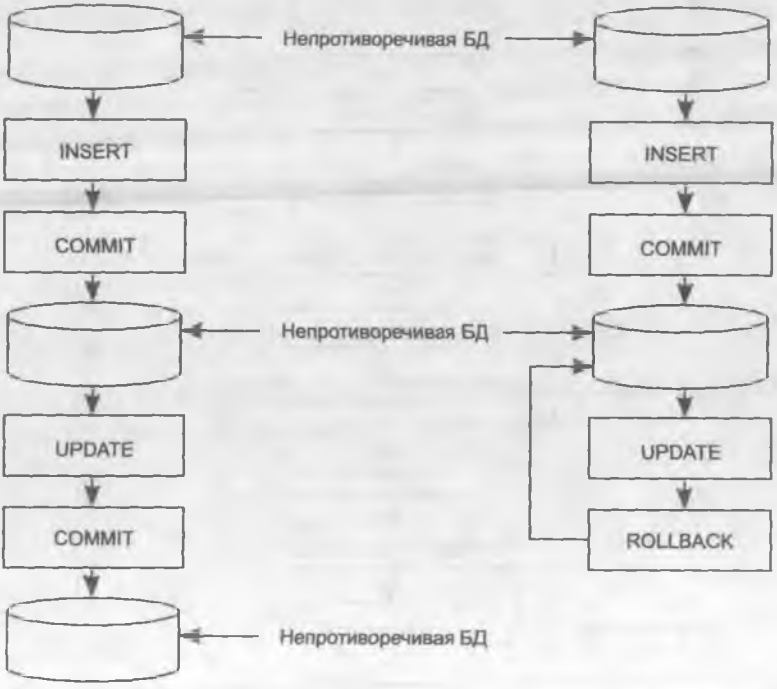


Рис. 12.2. Модель автоматического выполнения транзакций

12.1.2. Управляемое выполнение транзакций

Отличная от модели ANSI/ISO модель транзакций используется в СУБД Sybase, где применяется диалект Transact-SQL, в котором для обработки транзакций служат четыре инструкции (рис. 12.3):

- инструкция BEGIN TRANSACTION сообщает о начале транзакции, т. е. начало транзакции задается явно;
- инструкция COMMIT TRANSACTION сообщает об успешном выполнении транзакции, но при этом новая транзакция не начинается автоматически;
- инструкция SAVE TRANSACTION позволяет создать внутри транзакции *точку сохранения* и присвоить сохраненному состоянию *имя точки сохранения*, указанное в инструкции;
- инструкция ROLLBACK отменяет выполнение текущей транзакции и возвращает БД к состоянию, где была выполнена ин-

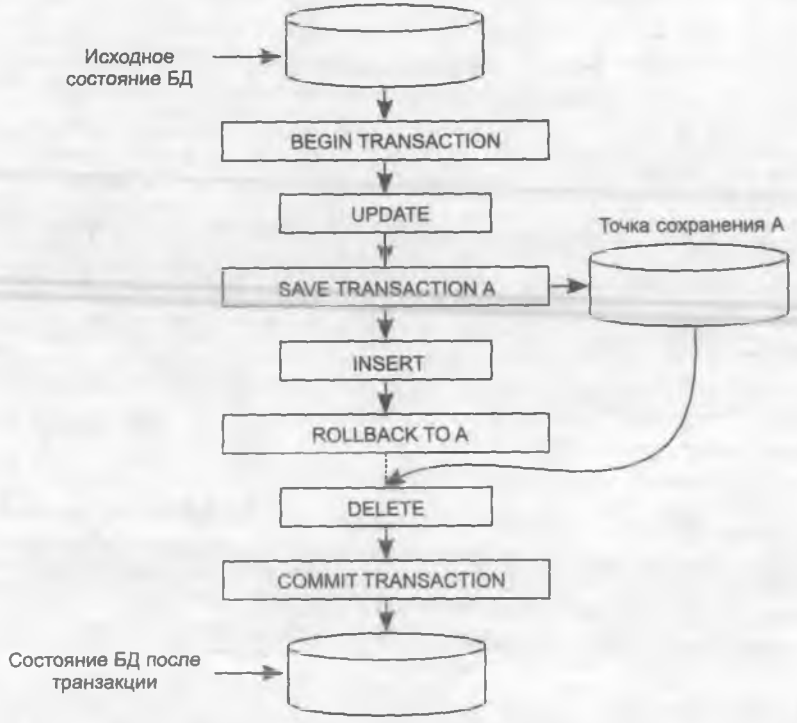


Рис. 12.3. Модель управляемого выполнения транзакций

струкция SAVE TRANSACTION (если в инструкции указана точка сохранения — ROLLBACK TO *имя_точки_сохранения*), или к состоянию начала транзакции.

12.2. Журнал транзакций

Возможность восстановления состояния базы данных после сбоев обеспечивается с помощью *журнала транзакций*. Журнализация изменений, т. е. сохранение во внешней памяти информации обо всех модификациях БД, тесно связана с управлением транзакциями.

Основным принципом согласованной политики записи изменений в журнал и непосредственно в базу данных является то, что запись об изменении объекта базы данных должна попадать во внешнюю память журнала раньше, чем измененный объект оказывается во внешней памяти базы данных. Соответствующий протокол журнализации (и управления буферизацией) называется Write Ahead Log (WAL) — «пиши сначала в журнал» и состоит в том, что если требуется сохранить во внешней памяти измененный объект базы данных, то перед этим нужно гарантировать сохранение во внешней памяти журнала записи о его изменении.

Другими словами, если во внешней памяти базы данных находится некоторый объект базы данных, по отношению к которому выполнена операция модификации, то во внешней памяти журнала обязательно находится запись, соответствующая этой операции. Каждая успешно завершившаяся транзакция должна быть реально зафиксирована во внешней памяти. Какой бы сбой не произошел, система должна иметь для восстановления состояния базы данных данные о результатах всех зафиксированных к моменту сбоя транзакций. Минимальным требованием, гарантирующим возможность восстановления последнего согласованного состояния базы данных, является сохранение во внешней памяти журнала всех записей об изменении базы данных отдельной транзакцией. При этом последней записью в журнал, производимой от имени транзакции, является специальная запись о конце транзакции.

Иногда для восстановления последнего согласованного состояния базы данных после сбоя одного журнала изменений базы данных недостаточно. Основой восстановления в этом случае являются журнал и *архивная копия* базы данных.

Восстановление начинается с обратного копирования базы данных из архивной копии. Затем для всех закончившихся транзакций в прямом смысле повторно выполняются все операции. Точнее, происходит следующее: по журналу в прямом направлении выполняются все операции; для транзакций, которые не закончились к моменту сбоя, выполняется откат (англ. back up).

Хотя к ведению журнала предъявляются особые требования по части надежности, реально возможна и его утрата. Тогда единственным способом восстановления базы данных является возврат к архивной копии. Конечно, в этом случае не удастся получить последнее согласованное состояние базы данных.

Рассмотрим способы производства архивных копий базы данных. Самый простой способ — архивировать базу данных при переполнении журнала. В этом случае запуск новых транзакций временно блокируется. Когда все транзакции закончатся и, следовательно, база данных придет в согласованное состояние, можно выполнять ее архивацию, после чего начинать заполнять журнал заново.

Можно выполнять архивацию базы данных реже, чем переполняется журнал. При переполнении журнала и окончании всех начатых транзакций можно архивировать сам журнал. Поскольку такой архивированный журнал, по сути дела, требуется только для воссоздания архивной копии базы данных, журнальная информация при архивации может быть существенно сжата.

В заключение сформулируем общие требования к системе восстановления данных в составе СУБД.

1. Пользователь не должен осуществлять рестарт транзакций или повторный ввод данных. Восстановление должно проходить на базе транзакций с помощью отмены или изменения отдельных транзакций.

2. Быстрое восстановление данных обеспечивается генерацией данных, используемых для восстановления.

3. При выполнении процедур автоматизированного восстановления пользователь не должен анализировать состав данных и выбирать сами процедуры.

Для восстановления базы данных СУБД имеют в своем составе сервисные программные средства.

Программы ведения системного журнала регистрируют операции над БД: описание соответствующей транзакции, код пользователя, текст входного сообщения, тип изменения БД, адреса изменяемых данных вместе с их значениями до и после изменения.

Программы архивации используются для регулярного получения копий БД для последующего ее восстановления.

Программы восстановления применяются для возврата БД или некоторых ее частей в состояние, предшествующее возникновению отказа. При этом используют архивную копию БД и системный журнал.

Программы отката ликвидируют последствия выполнения определенной транзакции в БД.

Программы записи контрольных точек и повторного исполнения позволяют ускорить восстановление.

12.3. Параллельное выполнение транзакций

При параллельной обработке данных (т. е. при совместной работе с БД нескольких пользователей) СУБД должна гарантировать, что пользователи не будут мешать друг другу. Средства обработки транзакций позволяют изолировать пользователей друг от друга таким образом, чтобы у каждого из них было ощущение монопольной работы с БД.

Транзакции являются подходящими единицами изолированности пользователей благодаря свойству сохранения целостности БД. Действительно, если с каждым сеансом работы с базой данных ассоциируется транзакция, то каждый пользователь начинает работу с согласованным состоянием базы данных, т. е. с таким состоянием, в котором база данных могла бы находиться, даже если бы пользователь работал с ней в одиночку.

Чтобы понять, как должны выполняться параллельные транзакции, рассмотрим проблемы, возникающие при параллельной работе с данными.

12.3.1. Пропавшие обновления

Рассмотрим пример работы двух диспетчеров с модифицированной БД «Сессия». Пусть Диспетчер 1 вносит в текущий учебный план для каждой дисциплины, читаемой на третьем курсе, сведения о преподавателях, параллельно изменяя при этом значение столбца *Нагрузка* в таблице «Кадровый_состав», а Диспетчер 2 выполняет такую же операцию для дисциплин второго курса.

Диспетчер 1 начинает работу по изменению таблицы «Учебный_план» для Дисциплины 1 с количеством часов, равным 50. В столбец *ID_Преподаватель* для этой дисциплины предполагается внести значение 5. Запрос текущей нагрузки преподавателя возвращает значение 350, и Диспетчер 1 подтверждает изменение таблицы «Учебный_план». При этом выполняются дополнительные действия по изменению столбца *Нагрузка* в таблице «Кадровый_состав» для строки с *ID_Преподаватель* = 5 (в столбец заносится значение 400).

До завершения операции Диспетчер 2 начинает те же действия для Дисциплины 2 с количеством часов 32, которую должен читать тот же преподаватель (*ID_Преподаватель* = 5). Запрос текущей нагрузки преподавателя также возвращает значение 350, с которым и работает дальше Диспетчер 2. Выполнив те же операции, что и Диспетчер 1 (но после него), Диспетчер 2 помещает в столбец *Нагрузка* значение 382, отменив тем самым предыдущие изменения (рис. 12.4).

Рис. 12.4. Проблема пропавшего изменения

Во избежание подобных ситуаций к СУБД по части синхронизации параллельно выполняемых транзакций предъявляется минимальное требование — отсутствие потери изменений.

12.3.2. Чтение «грязных» данных

Другой пример коллизий при несогласованной работе двух параллельных транзакций представлен на рис. 12.5.

Как показано на рисунке, Диспетчер 1 и Диспетчер 2 опять выполняют действия, описанные в предыдущем примере, но Диспетчер 2 начинает запрашивать данные о нагрузке преподавателя в тот момент, когда изменения, сделанные Диспетчером 1, уже зафиксированы в столбце *Нагрузка*, а транзакция еще не закончилась. Запрос Диспетчера 2 возвращает значение 400, и Диспетчер 2 вынужден отме-

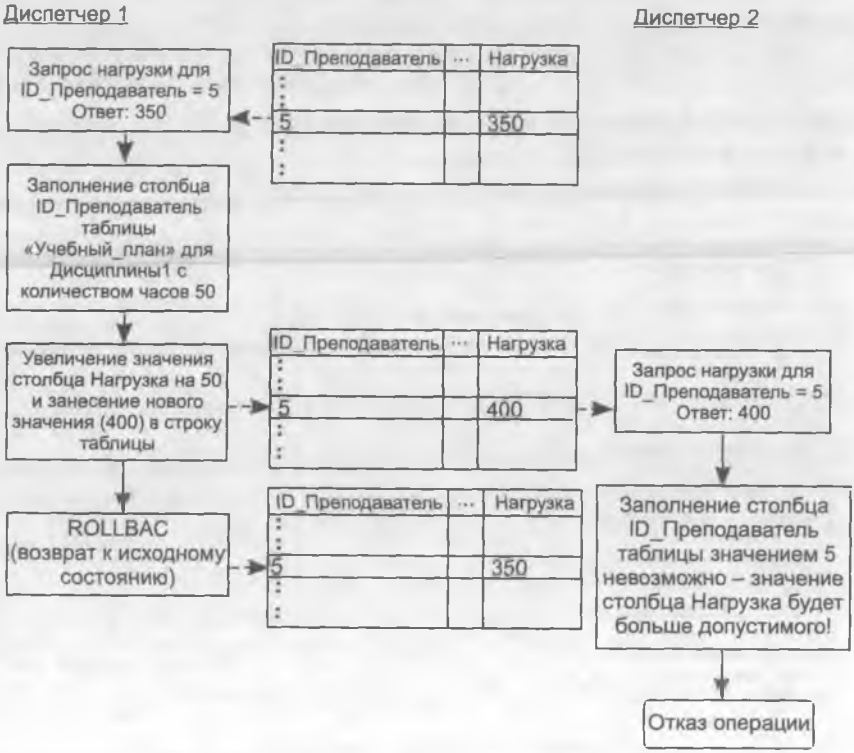


Рис. 12.5. Проблема чтения «грязных» данных

нить свои действия потому, что 400 часов — это максимальное разрешенное значение нагрузки. Между тем транзакция Диспетчера 1 закончилась возвратом к исходному состоянию, т. е. на самом деле Диспетчер 2 мог бы успешно завершить операцию.

Это тоже не соответствует требованию изолированности пользователей (каждый пользователь начинает свою транзакцию при согласованном состоянии базы данных и вправе ожидать увидеть согласованные данные). Чтобы избежать ситуации чтения «грязных» данных, необходимо, чтобы до завершения одной транзакции, изменившей некоторый объект, никакая другая транзакция не могла читать изменяемый объект.

12.3.3. Чтение несогласованных данных

Рассмотрим ситуацию, которая приводит к получению несогласованных данных при выполнении операций над БД (рис. 12.6).

По-прежнему Диспетчер 1 выполняет операцию по заполнению строки учебного плана, а Диспетчер 2 при выполнении своей операции должен сделать выбор между двумя преподавателями в соответствии с их текущей нагрузкой.

Начиная работу практически одновременно с Диспетчером 1, Диспетчер 2 получает следующие сведения: нагрузка первого препо-

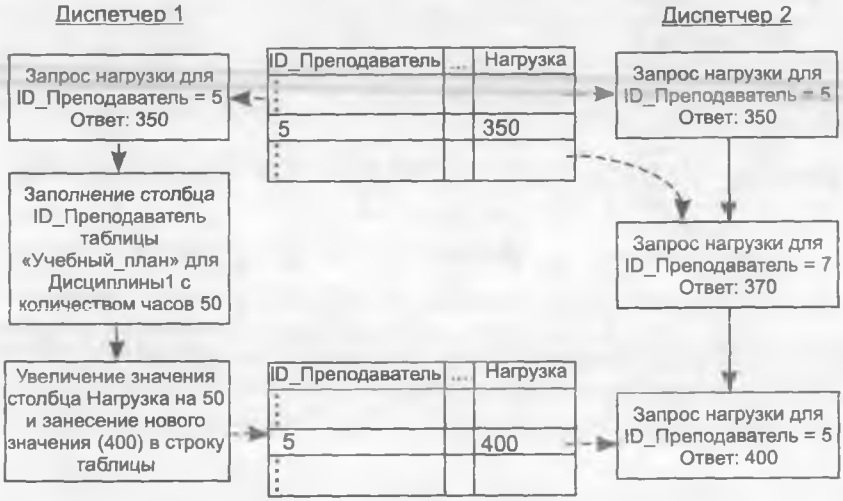


Рис. 12.6. Проблема чтения несогласованных данных

давателя (*ID_Преподаватель* = 5) составляет 350 часов, а нагрузка второго преподавателя (*ID_Преподаватель* = 7) составляет 370 часов. Далее Диспетчер 2 принимает решение в пользу первого преподавателя, но повторный запрос нагрузки возвращает значение 400, так как Диспетчер 1 уже сохранил новые данные в таблице «Кадровый_состав».

В большинстве систем обеспечение изолированности пользователей в подобных ситуациях является максимальным требованием к синхронизации транзакций.

12.3.4. Строки-призраки

К более тонким проблемам изолированности транзакций относится так называемая проблема строк-призраков, вызывающая ситуации, которые также противоречат изолированности пользователей. Рассмотрим следующий сценарий.

Диспетчер 1 инициирует Транзакцию 1, которая выполняет оператор выборки строк таблицы в соответствии с некоторым условием (например, формирование списка студентов, сдавших дисциплину с *ID_Дисциплина* = N, по таблице «Сводная_ведомость»). До завершения Транзакции 1 Транзакция 2, вызванная Диспетчером 2, вставляет в таблицу «Сводная_ведомость» новую строку, удовлетворяющую условию отбора Транзакции 1 (данные о результате сдачи дисциплины N еще одним студентом), и успешно завершается. При повторном выполнении Транзакцией 1 оператора выборки появляется строка, которая отсутствовала при первом выполнении оператора. Конечно, такая ситуация противоречит идее изолированности транзакций.

12.4. Сериализация транзакций

Чтобы добиться изолированности транзакций, СУБД должна использовать специальные методы регулирования совместного выполнения транзакций.

Метод *сериализации транзакций* — это механизм их выполнения по такому плану, когда результат совместного выполнения транзакций эквивалентен результату некоторого последовательного выполнения этих же транзакций. Обеспечение такого механизма является основной функцией управления транзакциями. Система, в которой

поддерживается метод сериализации транзакций, реально обеспечивает изолированность пользователей при работе с БД.

Основная реализационная проблема метода состоит в обеспечении такого выполнения транзакций, при котором не слишком ограничивалась бы их параллельность. Простейшим решением является их действительно последовательное выполнение, но существуют ситуации, когда можно выполнять операторы разных транзакций в любом порядке, и это не вызовет проблем. Примерами могут служить транзакции, выполняющие только операции чтения, или транзакции, работающие с разными объектами базы данных.

На самом деле между транзакциями могут существовать следующие виды конфликтов:

- Транзакция 2 пытается изменять объект, измененный незакончившейся Транзакцией 1 (W-W-конфликт);
- Транзакция 2 пытается изменять объект, прочитанный незакончившейся Транзакцией 1 (R-W-конфликт);
- Транзакция 2 пытается читать объект, измененный незакончившейся Транзакцией 1 (W-R-конфликт).

Практические методы сериализации транзакций основываются на учете и урегулировании этих конфликтов.

12.5. Захват и освобождение объекта

Для обеспечения сериализации транзакций применяются методы «захвата» и «освобождения» объектов, производимого по инициативе транзакции: транзакция «захватывает» объект, что приводит к его блокировке для других транзакций, и освобождает его только при своем завершении. При этом захваты объектов несколькими транзакциями на чтение совместимы (т. е. нескольким транзакциям разрешается читать один и тот же объект), захват объекта одной транзакцией на чтение несовместим с захватом другой транзакцией того же объекта на запись, и захваты одного объекта разными транзакциями на запись несовместимы. Тем самым выделяются два основных режима захватов:

- совместный режим — S (Shared), означающий разделяемый захват объекта и необходимый для выполнения операции чтения объекта;
- монопольный режим — X (eXclusive), означающий монопольный захват объекта и необходимый для выполнения операций записи, удаления и модификации.

Наиболее распространенным в СУБД, основанных на архитектуре «клиент — сервер», является подход, реализующий соблюдение *двухфазного протокола* захватов объектов БД. В общих чертах подход состоит в том, что перед выполнением любой операции над объектом базы данных транзакция запрашивает возможность захвата объекта в соответствующем режиме (в зависимости от вида операции — совместном или монопольном). В соответствии с протоколом выполнение транзакции разбивается на две фазы: *первая фаза* транзакции — накопление захватов; *вторая фаза* (фиксация или откат) — освобождение захватов.

При соблюдении двухфазного протокола основная проблема состоит в том, что следует считать объектом для захвата.

В контексте реляционных баз данных возможны следующие варианты:

- файл — физический (с точки зрения базы данных) объект, область хранения нескольких отношений и, возможно, индексов;
- таблица — логический объект, соответствующий множеству записей отношения;
- страница данных — физический объект, хранящий записи одного или нескольких отношений, индексную или служебную информацию;
- запись — элементарный физический объект базы данных.

Очевидно, что чем крупнее объект захвата, тем меньше захватов будет поддерживаться в системе и соответственно будут меньше накладные расходы. Более того, если выбрать в качестве уровня объектов для захватов файл или отношение, то будет решена даже проблема строк-призраков. Однако при использовании для захватов крупных объектов возрастает вероятность ожидания освобождения объекта и тем самым уменьшается степень параллельного выполнения транзакций. Фактически при укрупнении объекта синхронизационного захвата ситуация умышленно огрубляется и конфликты предполагаются в тех ситуациях, когда на самом деле конфликтов нет.

Таким образом, можно резюмировать, что транзакция — это законченный блок обращений к базе данных и некоторых действий над ней, для которого гарантируется выполнение четырех условий ACID (Atomicity, Consistency, Isolation, Durability):

- *атомарность* — операции транзакции образуют неразделимый атомарный блок с определенным началом и концом. Этот блок либо выполняется от начала до конца, либо не выполняется во-

обще. Если в процессе выполнения транзакции произошел сбой, происходит откат к исходному состоянию;

- *согласованность* — по завершении транзакции все задействованные объекты находятся в согласованном состоянии;
- *изолированность* — одновременный доступ транзакций различных приложений к разделяемым объектам координируется таким образом, чтобы эти транзакции не влияли друг на друга;
- *долговременность* — все изменения данных, осуществленные в процессе выполнения транзакции, не могут быть потеряны.

Контрольные вопросы и задания

1. Дайте определение транзакции.
2. Охарактеризуйте модели автоматического и управляемого выполнения транзакций.
3. Назовите виды конфликтов при параллельном выполнении транзакций.
4. Что такое сериализация транзакций?
5. Охарактеризуйте методы «захвата» и «освобождения» объектов.
6. Назовите основные режимы «захвата» объектов.
7. Что такое журнал транзакций?
8. Перечислите основные сервисные программные средства восстановления базы данных в составе СУБД.

Глава 13
УПРАВЛЕНИЕ БАЗАМИ ДАННЫХ В СУБД

Для обеспечения эффективного контролируемого управления доступом к данным, целостности и сокращения избыточности хранимых данных большинство СУБД должны быть тесно связаны с операционной системой: многопользовательские приложения, обработка распределенных запросов, защита данных, использование многопроцессорных систем и мультипоточных технологий требуют ресурсов, управление которыми обычно является функцией ОС. В частности, средства управления доступом и обеспечения защиты обычно интегрируются с соответствующими средствами операционной системы.

Напомним, что с точки зрения операционной системы база данных — это один или несколько обычных файлов ОС, доступ к которым осуществляется не напрямую, а через СУБД. При этом данные обычно располагаются на машине-сервере, а СУБД обеспечивает корректную и эффективную их обработку из приложений, выполняющихся на машинах-клиентах.

С другой стороны, в рамках СУБД база данных — это логически структурированный набор объектов, связанных не только с хранением и обработкой прикладных данных, но и обеспечивающих целостность БД, управление доступом, представление данных и т. д. Например, в MS SQL Server база данных включает следующие объекты:

- таблицы;
- хранимые процедуры;
- триггеры;
- представления;
- правила;
- пользовательские типы данных;

- индексы;
- пользователи;
- роли;
- публикации;
- диаграммы.

Кроме того, при создании базы данных для нее всегда определяется *журнал транзакций*, который используется для восстановления состояния базы данных в случае сбоев или потери данных. Журнал размещается в одном или нескольких файлах. В журнале регистрируются все транзакции и все изменения, произведенные в их рамках. Транзакция не считается завершенной, пока соответствующая запись не будет внесена в журнал.

Управление базой данных производится обычно с помощью нескольких сервисных программ — отдельных приложений, выполняемых в среде операционной системы. Рассмотрим основные функции и компоненты управления на примере сервера реляционных баз данных MS SQL Server.

Бóльшая часть функций администрирования работы пользователей, серверов и баз данных сосредоточена в специальном приложении, которое позволяет осуществлять, в частности, следующие функции:

- запускать и конфигурировать SQL Server;
- управлять доступом пользователей к объектам БД;
- создавать и модифицировать базы данных и их объекты, такие как таблицы, индексы, представления и т. д.;
- управлять выполнением заданий «по расписанию»;
- управлять репликациями;
- создавать резервные копии баз данных и журналов транзакций.

13.1. Планирование БД

Использование концепции файлов и файловых групп для физического размещения хранимых данных упрощает управление базами данных и дисковой памятью, а также обеспечивает гибкость при размещении конкретных объектов на устройстве или устройствах. Причем в этом случае обеспечивается реальное распределение данных между всеми входящими в группу файлами (отдельными дисковыми устройствами или RAID-массивами): дисковые устройства действительно одновременно, а не поочередно будут заполняться поступаю-

щими данными, поскольку данные будут пропорционально «чередоваться» по файлам группы (рис. 13.1).

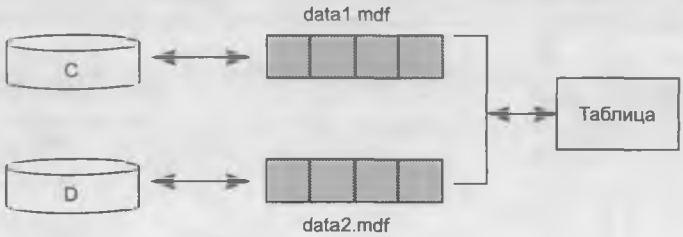


Рис. 13.1. Чередование данных в файловой группе

Для повышения производительности системы в ряде случаев может использоваться индекс — отдельная физическая структура в базе данных, создаваемая на основе таблицы и предназначенная для ускорения выборки данных, поиск которых осуществляется по значению из проиндексированного столбца. Кроме того, индексы используются для обеспечения уникальности строк и столбцов таблиц, упорядочения данных таблицы в отдельном файле или группе файлов для повышения скорости доступа.

Однако наличие индекса замедляет такие операции с таблицей, как вставка, обновление и удаление данных: индексы являются *динамически поддерживаемыми* структурами, т. е. при вставке, удалении или обновлении данных информация в индексах также должна быть изменена для отражения выполненных в таблице изменений. Для такой обработки требуются дополнительные операции ввода-вывода.

Кластерный индекс представляет собой двоичное дерево, в котором на нулевом уровне (уровне листьев) содержатся страницы актуальных данных таблицы, а физическое расположение информации в данном индексе логически упорядочено. Такое размещение данных позволяет сократить время доступа к данным, но только при отборе по этому индексу. В других случаях это приводит к задержкам, так как доступ к данным осуществляется только через индекс и начинается всегда с корня.

Для отдельной таблицы можно построить только один кластерный индекс.

В случае *некластерных индексов* страницы уровня листа содержат не текущие данные таблицы (как в случае кластерного индекса), а указатель на строку данных, включающий номер страницы данных и порядковый номер записи на странице.

Некластерный индекс позволяет быстро получить доступ к данным и не требует физического переупорядочения строк данных таблицы (рис. 13.2).

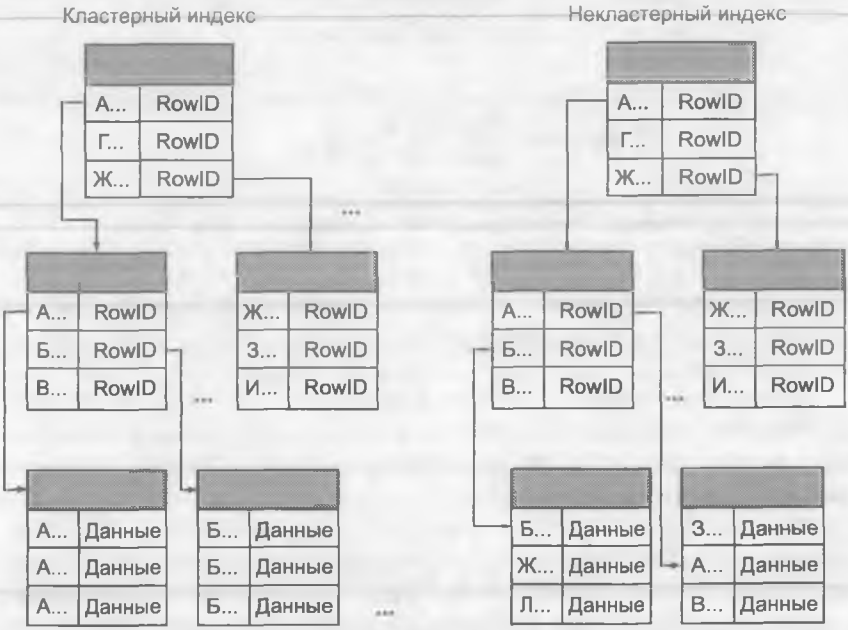


Рис. 13.2. Кластерный и некластерный индексы

13.2. Управление доступом

Система безопасности SQL Server имеет несколько уровней безопасности:

- операционная система;
- SQL Server;
- база данных,
- объект базы данных.

С другой стороны, механизм безопасности предполагает существование четырех типов пользователей:

- системный администратор, имеющий неограниченный доступ;
- владелец БД, имеющий полный доступ ко всем объектам БД;
- владелец объектов БД;

- другие пользователи, которые должны получать разрешение на доступ к объектам БД.

Модель безопасности SQL Server включает следующие компоненты:

- тип подключения к SQL Server;
- пользователь базы данных;
- пользователь (guest);
- роли (roles).

13.2.1. Тип подключения к SQL Server

При подключении (и в зависимости от типа подключения) SQL Server поддерживает два режима безопасности:

- режим аутентификации Windows NT;
- смешанный режим аутентификации.

В ***режиме аутентификации Windows NT*** используется система безопасности Windows NT и ее механизм учетных записей. Этот режим позволяет SQL Server использовать имя пользователя и пароль, которые определены в Windows, и тем самым обходить процесс подключения к SQL Server. Таким образом, пользователи, имеющие действующую учетную запись Windows, могут подключиться к SQL Server, не сообщая своего имени и пароля. Когда пользователь обращается к СУБД, последняя получает информацию об имени пользователя и пароле из атрибутов системы сетевой безопасности пользователей Windows (которые устанавливаются, когда пользователь подключается к Windows).

В ***смешанном режиме аутентификации*** задействованы обе системы аутентификации: Windows и SQL Server. При использовании системы аутентификации SQL Server отдельный пользователь, подключающийся к SQL Server, должен сообщить имя пользователя и пароль, которые будут сравниваться с хранимыми в системной таблице сервера.

13.2.2. Пользователи базы данных

Понятие *пользователь базы данных* относится к базе (или базам) данных, к которым может получить доступ отдельный пользователь. После успешного подключения сервер определяет, имеет ли этот

пользователь разрешение на работу с базой данных, к которой обращается.

Единственным исключением из этого правила является пользователь guest (гость). Особое имя пользователя guest разрешает любому подключившемуся к SQL Server пользователю получить доступ к этой базе данных. Пользователю с именем guest назначена роль public.

Права доступа

Для управления правами доступа в SQL Server используются следующие опции:

- GRANT. Позволяет выполнять действия с объектом, а для команды — выполнять ее;
- REVOKE. Аннулирует права доступа для объекта, а для команды — не позволяет выполнить ее;
- DENY. Не разрешает выполнять действия с объектом (в то время как REVOKE просто удаляет эти права доступа).

Объектные права доступа позволяют контролировать доступ к объектам в SQL Server, предоставляя и аннулируя права доступа для таблиц, столбцов, представлений и хранимых процедур. Чтобы выполнить по отношению к некоторому объекту некоторое действие, пользователь должен иметь соответствующее право доступа. Например, если пользователь хочет выполнить оператор SELECT * FROM table, то он должен иметь права выполнения оператора SELECT для таблицы table.

Командные права доступа определяют тех пользователей, которые могут выполнять административные действия, например создавать или копировать базу данных. Ниже приведены командные права доступа:

CREATE DATABASE — право создания базы данных;

CREATE DEFAULT — право создания стандартного значения для столбца таблицы;

CREATE PROCEDURE — право создания хранимой процедуры.

CREATE ROLE — право создания правила для столбца таблицы;

CREATE TABLE — право создания таблицы;

CREATE VIEW — право создания представления;

BACKUP DATABASE — право создания резервной копии;

BACKUP TRANSACTION — право создания резервной копии журнала транзакций.

13.2.3. Роли

Назначение пользователю некоторой роли позволяет ему выполнять все функции, разрешенные этой ролью. По сути роли логически группируют пользователей, имеющих одинаковые права доступа. В SQL Server есть следующие типы ролей:

- роли уровня сервера;
- роли уровня базы данных.

Роли уровня сервера

С помощью этих ролей предоставляются различные степени доступа к операциям и задачам сервера. Роли уровня сервера заранее определены и действуют в пределах сервера. Они не зависят от конкретных баз данных, и их нельзя модифицировать.

В SQL Server существуют следующие типы ролей уровня сервера:

- Sysadmin — дает право выполнить любое действие в SQL Server;
- Serveradmin — дает право изменить параметры SQL Server и завершить его работу;
- Setupadmin ль— дает право инсталлировать систему репликации и управлять выполнением расширенных хранимых процедур;
- Securityadmin — дает право контролировать параметры учетных записей для подключения к серверу и предоставлять права доступа к базам данных;
- Processadmin — дает право управлять ходом выполнения процессов в SQL Server;
- Dbcreator — дает право создавать и модифицировать базы данных;
- Diskadmin — дает право управлять файлами баз данных на диске.

Роли уровня базы данных

Роли уровня базы данных позволяют назначить права для работы с конкретной базой данных отдельному пользователю или группе. Роли уровня базы данных можно назначать учетным записям пользователей в режиме аутентификации Windows или SQL Server. Роли могут быть и вложенными, так что учетным записям можно назначить иерархическую группу прав доступа.

В SQL Server существует три типа ролей:

- заранее определенные роли;
- определяемые пользователем роли;
- неявные роли.

Заранее определенными являются стандартные роли уровня БД. Эти роли имеет каждая база данных SQL Server.

Заранее определенные роли зависят от конкретной базы данных и не могут быть изменены. Ниже перечислены стандартные роли уровня базы данных:

db_owner — определяет полный доступ ко всем объектам базы данных, может удалять и воссоздавать объекты, а также присваивать объектные права другим пользователям. Охватывает все функции, перечисленные ниже для других стандартных ролей уровня базы данных;

db_accessadmin — осуществляет контроль доступа к базе данных путем добавления или удаления пользователей в режимах аутентификации;

db_datareader — определяет полный доступ к выборке данных (с помощью оператора SELECT) из любой таблицы базы данных. Запрещает выполнение операторов INSERT, DELETE и UPDATE для любой таблицы БД;

db_datawriter — разрешает выполнять операторы INSERT, DELETE и UPDATE для любой таблицы базы данных. Запрещает выполнение оператора SELECT для любой таблицы базы данных;

db_ddladmin — дает возможность создавать, модифицировать и удалять объекты базы данных;

db_securityadmin — управляет системой безопасности базы данных, а также назначением объектных и командных разрешений и ролей для базы данных;

db_backupoperator — позволяет создавать резервные копии базы данных;

db_denydatareader — не разрешает выполнение оператора SELECT для всех таблиц базы данных. Позволяет пользователям изменять существующие структуры таблиц, но не позволяет создавать или удалять существующие таблицы;

db_denydatawriter — не разрешает выполнение операторов модификации данных (INSERT, DELETE и UPDATE) для любых таблиц базы данных;

public — автоматически назначаемая роль сразу после предоставления права доступа пользователя к БД.

Роли, определяемые пользователем, позволяют группировать пользователей и назначать каждой группе конкретную функцию безопасности.

Существуют два типа ролей уровня базы данных, определяемых пользователем:

- стандартная роль;
- роль уровня приложения.

Стандартная роль предоставляет зависящий от базы данных метод создания определяемых пользователем ролей. Самое распространенное назначение стандартной роли — логически сгруппировать пользователей в соответствии с их правами доступа. Например, в приложениях выделяют несколько типов уровней безопасности, ассоциируемых с тремя категориями пользователей. *Опытный пользователь* может выполнять в базе данных любые операции; *обычный пользователь* может модифицировать некоторые типы данных и обновлять данные; *неквалифицированному пользователю* обычно запрещается модифицировать любые типы данных.

Роль уровня приложения позволяет пользователю выполнять права некоторой роли. Когда пользователь принимает роль уровня приложения, он берет на себя выполнение новой роли и временно отказывается от всех других назначенных ему прав доступа к конкретной базе данных. Роль уровня приложения имеет смысл применять в среде, где пользователи делают запросы и модифицируют данные с помощью клиентского приложения.

13.3. Управление обработкой. Представления, хранимые процедуры, триггеры

Для решения типовых (часто повторяющихся) задач выборки или обновления данных, а также в значительной части для управления доступом к данным (как альтернатива механизму разрешения — запрета) и обеспечения целостности данных целесообразно использовать процедуры. Кроме того, другое преимущество, уже в части администрирования, состоит в том, что не надо специально определять пользователю права доступа к таблицам и представлениям, используемым в процедуре: достаточно определить только разрешение на выполнение процедуры.

Существуют два способа взаимодействия приложения с SQL Server. Можно создать приложение, отправляющее клиентские операторы T-SQL на сервер, либо создать хранимые процедуры непосредственно на сервере. В первом случае операторы каждый раз рекомпи-

лируются сервером. Второй способ активизирует хранимые процедуры, вызывая их из приложения одним оператором. При первом вызове хранимой процедуры она компилируется и создается план ее выполнения, который сохраняется в памяти. При последующих вызовах SQL Server будет использовать этот план и процедуру повторно не компилирует. Таким образом, когда для решения определенных задач требуется многократно выполнить одну и ту же последовательность операторов SQL, применение хранимой процедуры обеспечивает более высокую производительность.

Для управления обработкой в процедурах можно использовать локальные переменные, которые создаются с помощью оператора DECLARE. Переменная доступна с момента ее объявления и до выхода из процедуры. После выхода из процедуры на переменную ссылаться нельзя. Локальные переменные можно объявлять в пакете, в сценарии, внешней программе, а также в хранимой процедуре. В операторе DECLARE необходимо указать имя переменной и ее тип.

13.3.1. Представления

Представления (*View*) существуют независимо от информации в базе данных, но тесно с ней связаны. Представления используются для фильтрования и предварительной обработки данных.

Представление — это по существу некая виртуальная таблица, содержащая результаты выполнения запроса (оператора SELECT) к одной или нескольким таблицам. Для конечного пользователя представление выглядит как обычная таблица в базе данных, над которой можно выполнять операторы SELECT, INSERT, UPDATE и DELETE. В действительности представление хранится в виде предопределенного оператора SQL.

Различные типы представлений имеют свои преимущества и недостатки. Выбор того или иного типа представлений полностью зависит от задач приложения. Выделяют следующие типы представлений:

- *подмножество полей таблицы* — состоит из одного или более полей таблицы и считается самым простым типом представления. Обычно используется для упрощения представления данных и обеспечения безопасности;
- *подмножество записей таблицы* — включает определенное количество записей таблицы и также применяется для обеспечения безопасности;

- *соединение двух и более таблиц* — создается соединением нескольких таблиц и используется для упрощения сложных операций соединения;
- *агрегирование информации* — создается группированием данных и также применяется для упрощения сложных операций.

Еще одно преимущество представлений заключается в том, что они могут иметь более низкий уровень безопасности, чем их исходные таблицы. Запрос для представления выполняется согласно уровню безопасности вызывающего его пользователя. Таким образом, представление можно применять для сокрытия данных от определенной группы пользователей.

Представления, как и индексы, можно создавать различными способами: использовать для этого «мастер» или команду T-SQL, имеющую в общем случае следующий формат:

CREATE VIEW *имя_представления [столбец[,..]]*
AS *SELECT-оператор*

Следует отметить, что использование в операторе SELECT предложения WHERE позволяет локализовать доступ пользователя к данным даже на уровне отдельных строк и столбцов.

13.3.2. Хранимые процедуры

Хранимая процедура (*stored procedure*) — это набор операторов T-SQL, которые SQL SERVER компилирует в единый план выполнения. Этот план сохраняется в кэше процедур при первом выполнении хранимой процедуры, и затем план можно повторно использовать уже без рекомпиляции при каждом вызове. Хранимая процедура аналогична процедурам в языках программирования: она может принимать входные параметры, возвращать данные и коды завершения.

Применение хранимых процедур улучшает производительность, например при использовании в хранимой процедуре условных операторов (таких как IF и WHILE), поскольку условие будет проверяться непосредственно на сервере и серверу не потребуется возвращать промежуточные результаты проверки условия программам-клиентам.

Хранимые процедуры также позволяют централизованно контролировать выполнение задачи, что гарантирует соблюдение бизнес-правил.

Хранимые процедуры, как и представления, можно создавать различными способами: использовать для этого «мастер» или команду T-SQL, имеющую в общем случае следующий формат:

```
CREATE PROCEDURE имя_процедуры [(%параметр1 тип_данных[...]]
AS SQL-операторы
```

Существует два типа хранимых процедур: системные и пользовательские. Первые поддерживается SQL Server и применяются для управления сервером и отображения информации о базах данных и пользователях. Вторые создаются пользователями для выполнения прикладных задач.

13.3.3. Триггеры

Триггер (*trigger*) — это особый тип хранимой процедуры, которая автоматически выполняется при изменении таблицы с помощью операторов UPDATE, INSERT или DELETE. Как и хранимые процедуры, триггеры содержат операторы T-SQL, но в отличие от процедур запускаются не индивидуально, а автоматически при выполнении операций изменения данных. Триггеры наряду с ограничениями обеспечивают целостность данных и соблюдение бизнес-правил, однако их следует использовать разумно. Например, не нужно создавать триггер, проверяющий наличие значения первичного ключа в одной таблице, чтобы определить, можно ли вставить значение в соответствующее поле другой таблицы. Однако трудно обойтись без триггеров при выполнении каскадных изменений в дочерних таблицах.

Триггер создается на одной таблице в текущей базе данных, хотя может использовать данные других таблиц и объекты других баз данных. Триггеры нельзя создавать на представлениях, временных и системных таблицах. Таблица, для которой определен триггер, называется *таблицей триггера*.

Существуют три типа триггеров: UPDATE, INSERT и DELETE, каждый из которых инициируется при выполнении одноименной команды. Операции UPDATE, INSERT и DELETE иногда называют событиями изменения данных. Можно создать триггер, который будет срабатывать при возникновении более чем одного события, например запускаться в ответ на операторы UPDATE или INSERT. Такие комбинированные триггеры называются UPDATE/INSERT. Возможно создание триггеров, срабатывающих при выполнении любого

из трех операторов обновления данных (триггер UPDATE/INSERT/DELETE).

Триггеры, как и представления, можно создавать различными способами: использовать для этого «мастер» или команду T-SQL, имеющую в общем случае следующий формат:

```
CREATE TRIGGER имя_триггера
ON имя_таблицы
FOR [INSERT] [,] [UPDATE] [,] [DELETE]
AS SQL-операторы
```

В программе-триггере нельзя использовать операторы создания, реструктуризации, удаления объектов, реконфигурации и восстановления.

Работа триггеров подчиняется следующим правилам:

- триггеры запускаются только после завершения выполнения вызывающего их оператора. Например, триггер UPDATE не начинает работать, пока не завершится выполнение оператора UPDATE;
- триггер не начинает работать, если при выполнении оператора происходит нарушение какого-либо ограничения таблицы или возникает другая ошибка;
- триггер и вызывающий его оператор образуют транзакцию. В результате вызова из триггера оператора ROLLBACK отменяются изменения, выполненные триггером и оператором. При возникновении серьезной ошибки, например при отключении пользователя, SQL-Server автоматически выполнит откат всей транзакции;
- триггер запускается один раз для каждого оператора, независимо от количества изменяемых оператором записей.

Триггеры возвращают результаты своей работы в приложение, подобно хранимым процедурам. Как правило, пользователь не ожидает вывода после выполнения операторов UPDATE, INSERT и DELETE, вызывающих срабатывание триггеров. Если триггер возвращает данные, приложение должно содержать код, правильно интерпретирующий результаты модификации таблицы и вывод триггера.

Для каждого триггера SQL Server создает две временные таблицы, на которые можно ссылаться в описании триггера. Эти таблицы хранятся в памяти и локальны по отношению к триггеру, то есть триггер имеет доступ только к своей собственной версии таблиц. Временные

таблицы применяются для сравнения состояния таблицы до и после внесения изменений.

В MS SQL Server возможно создание нескольких триггеров на таблице для каждого события изменения данных (UPDATE, INSERT или DELETE) и рекурсивный вызов триггера. Например, если для таблицы уже определен триггер UPDATE, то можно определить еще один триггер UPDATE для той же самой таблицы. В этом случае после выполнения соответствующего оператора сработают оба триггера. Кроме того, допускаются вложенные триггеры, которые срабатывают в результате выполнения других триггеров. Они отличаются от рекурсивных тем, что не запускают сами себя.

13.4. Управление транзакциями

Репликация базы данных заключается в копировании, или *тиражировании*, данных из одной таблицы или базы данных в другую.

Предприятие с сетью региональных отделений — показательный случай для использования системы с репликацией транзакций. Каждый филиал ведет работу со своими счетами, информация о которых содержится в его базе данных. Главный офис является подписчиком на базы данных всех филиалов, что позволяет собирать в нем информацию по всей организации.

Репликация основана на метафоре «издатель — подписчик»: *издатель (публикующий сервер)* предоставляет данные: *распространитель* содержит тиражируемую базу или служебную информацию для управления репликацией; *подписчик* — получает и обрабатывает реплицированные данные. Данные передаются от публикующего или рассылающего сервера в направлении каждого из подписчиков. Данные не могут пересылаться подписчику непосредственно от другого подписчика. Если один из подписчиков перестает функционировать, это не должно оказывать никакого влияния на издателя или других подписчиков.

В схеме репликации транзакций для доставки данных от публикующего сервера на каждый из серверов-подписчиков используются три следующих компонента:

- *агент синхронизации* (Snapshot Agent). Создает файлы данных и структуры, используемые для согласования новых подписчиков с текущим состоянием публикации;

- *агент чтения журнала* (Log Reader Agent). Считывает из журнала транзакций публикующего сервера подлежащие репликации транзакции и помещает их в базу данных рассылки;
- *агент рассылки* (Distributation Agent). Пересылает подлежащие репликации транзакции из базы данных рассылки всем подписчикам на публикацию.

Каждая публикация (набор реплицируемых данных — *статей*, которыми могут быть таблицы, записи, поля или хранимые процедуры) создается для выделения данных, подлежащих репликации в базе данных подписчиков. Агент синхронизации создает файл схемы, предназначенный для создания в базе данных табличных структур реплицируемых данных. Этот агент также создает файлы, содержащие данные из публикуемых статей, и обновляет содержимое базы данных рассылки для фиксации нового задания на согласование.

В журнале транзакций публикующего сервера все транзакции, подлежащие репликации в адрес одного или более подписчиков, помечаются специальным флажком. Агент чтения журнала считывает из журнала все помеченные этим флажком команды INSERT, UPDATE и DELETE. Агент следит за появлением подлежащих репликации транзакций для каждой публикации, существующей в базе данных публикующего сервера. Любая обнаруженная транзакция копируется им в базу данных рассылки. Затем агент чтения журналов адресует рассылаемые данные каждому подписчику на публикацию.

После исходного согласования состояний подписчика и публикующего сервера весь объем данных никогда не пересылается в адрес подписчика. Поддержание актуального состояния базы данных подписчика обеспечивается агентом рассылки. Он пересылает подписчику все команды INSERT, UPDATE и DELETE, введенные пользователями на стороне публикующего сервера. Очень важно четко представлять всю последовательность действий, которые выполняются при передаче подписчику сведений об изменениях, проведенных на стороне публикующего сервера.

При создании публикации разработчик может разрешить подписчику выполнять обновление собственной локальной копии данных. В этом случае все выполненные на стороне такого подписчика изменения будут переданы в обратном направлении на публикующий сервер, а последний разошлет их в адрес всех остальных серверов-подписчиков. Отсутствие конфликтов и гарантия внесения изменений на публикующий сервер обеспечиваются благодаря использованию на сервере-подписчике протокола двухступенчатой фиксации измене-

ний. Если публикующий сервер по какой-либо причине не сможет получить сведения о внесенных изменениях, выполненная на стороне подписчика транзакция будет отменена и восстановится исходное состояние базы данных. В такой схеме *непосредственно обновляемых подписчиков* используются триггеры, хранимые процедуры, координатор распределенных транзакций, а также средства обнаружения конфликтов и рекурсии.

Триггеры размещаются на стороне подписчика. Это гарантирует, что любая начатая на сервере-подписчике транзакция будет обязательно зафиксирована на публикующем сервере, прежде чем появится возможность зафиксировать ее на сервере-подписчике. Здесь используется протокол с двухступенчатой фиксацией изменений (Two Phase Commit — 2PC). Если транзакцию не удастся зафиксировать на публикующем сервере, она будет отменена и на сервере-подписчике, поэтому состояние обеих баз данных останется согласованным.

Хранимые процедуры размещаются на стороне публикующего сервера. Это гарантирует, что любые реплицируемые транзакции будут выполнены только в том случае, если это не приведет к конфликту. Если в результате выполнения транзакции возникает конфликт, на серверах обоих узлов для данной транзакции будет выполнен откат.

Координацию выполнения двухступенчатой фиксации изменений между публикующим сервером и сервером-подписчиком осуществляет *Координатор распределенных транзакций* (Microsoft Distributed Transaction Coordinator — MS DTC), который вызывает выполнение удаленных хранимых процедур.

13.5. Резервное копирование и восстановление

Архивирование и восстановление базы данных с корректировкой целостности основаны на механизме регистрации изменений, использующем журнал транзакций и контрольные точки.

В журнале транзакций регистрируются все транзакции и все изменения базы данных, произведенные в их рамках. Транзакция не считается завершенной, пока соответствующая запись не будет внесена в журнал.

Журнал может размещаться в нескольких файлах, допускающих автоматический рост. Журнал рассматривается не как таблица, а как отдельный файл в базе данных: запись в журнал ведется блоками лю-

бого размера, не зависящего от размера страниц сервера. При обновлении журнала или его архивировании происходит усечение журнала.

Контрольная точка — это операция согласования состояния базы данных в физических файлах с текущим состоянием кэша — системного буфера. С целью улучшения производительности сохраняемые в БД данные сначала помещаются в кэш, а потом система перезапишет модифицированные страницы на диск (*отложенная запись*), причем пользователь не может знать, когда эта запись производится.

Контрольная точка выполняется командой CHECKPOINT при завершении работы сервера, а также в соответствии с установленным интервалом контрольных точек и включает выполнение следующих операций:

- запись на диск всех страниц, измененных к началу контрольной точки;
- запись в журнал транзакций списка незавершенных транзакций;
- запись в журнал транзакций всех измененных страниц;
- регистрация завершения контрольной точки в базе данных (а не в журнале транзакций).

Резервное копирование выполняется для каждой базы индивидуально и может производиться несколькими способами.

Полное резервное копирование обеспечивает архивирование всех данных базы, размещенных как в группах файлов, так и в отдельных файлах. Этот способ наиболее часто используется для архивирования баз данных не очень большого размера. В противном случае надо использовать выборочное копирование или копирование групп файлов.

Выборочное (дифференциальное) резервное копирование обеспечивает архивирование только тех данных базы, которые были изменены с момента последнего архивирования.

Резервное копирование журнала транзакций обеспечивает архивирование и усечение журнала.

В случае *резервного копирования файлов и групп файлов* их можно копировать вместе или по отдельности. Полностью восстановить базу данных с помощью резервной копии файлов и группы файлов несколько сложнее, чем с помощью обычной резервной копии. Для восстановления таблиц и индексов, которые охватывают несколько групп файлов, нужно, чтобы эти файлы и группы файлов были скопированы вместе с охватывающими их объектами.

Для правильного восстановления базы данных на основе файлов или группы файлов, необходимо использовать резервную копию журнала транзакций.

Для размещения архивных копий должно быть создано логическое устройство (которое может быть и отдельным физическим устройством).

Информация о выполнении резервного копирования сохраняется как запись в системной таблице базы, что позволяет определить, когда и на какое устройство сделана копия.

Процесс восстановления базы данных зависит от типа архива. При восстановлении из дифференциального архива или из архива журнала транзакций необходимо предварительно восстановить БД из последнего полного архива.

Контрольные вопросы и задания

1. Определите понятие «база данных» в рамках СУБД.
2. В чем состоит сходство и различие кластеризованного и некластеризованного индексов?
3. Какие компоненты включает в себя модель безопасности?
4. Когда нужно использовать систему аутентификации Windows NT и SQL Server?
5. Дайте сравнительный анализ типов ролей уровня сервера, уровня базы данных, уровня приложений.
6. Каковы назначение и типы «ролей»?
7. Назначение хранимых процедур и триггеров. В чем состоит сходство и различие хранимых процедур и триггеров?
8. Использование «представлений» для управления доступом.
9. Назначение и обобщенная схема репликации баз данных.
10. Назначение и использование «контрольных точек» для восстановления БД.
11. Назначение и основные способы резервного копирования.

Литература

1. *Аткинсон М., Бансилон Ф., ДеВитт Д.* и др. Манифест систем объектно-ориентированных баз данных // СУБД. 1995. № 4.

2. *Вирт Н.* Алгоритмы и структуры данных / пер. с англ. М.: Мир, 1989.

3. *Грофф Дж, Вайнберг П.* SQL: полное руководство / пер. с англ. 2-е изд. К.: BHV, 2001.

4. *Дейт К.Дж.* Введение в системы баз данных / пер. с англ. 8-е изд. М.: Вильямс, 2006.

5. *Дунаев С.Б.* Доступ к базам данных и техника работы в сети. Практические приемы современного программирования. М.: ДИАЛОГ-МИФИ, 1999.

6. *Зиндер Е.З.* Проектирование баз данных: новые требования, новые подходы // СУБД. 1996. № 3.

7. *Карпова Т.С.* Базы данных: модели, разработка, реализация. СПб.: Питер, 2001.

8. *Ким Вон.* Технология объектно-ориентированных баз данных // Открытые системы. 1994. № 4.

9. *Когаловский М.Р.* Абстракции и модели в системах баз данных // Открытые системы. 1998. № 4—5.

10. *Коннолли Т., Бегг К.* Базы данных. Проектирование, реализация и сопровождение. Теория и практика / пер. с англ. 3-е изд. М.: Вильямс, 2003.

11. *Кузнецов С.Д.* Основы баз данных. Курс лекций. М.: Интернет-университет информационных технологий, 2005.

12. *Мартин Дж.* Организация баз данных в вычислительных системах. М.: Мир, 1980.

13. *Мартин Дж.* Превратите вашу компанию в киберкорпорацию // Computerworld Россия. 1995. 14 нояб.

14. *Мирошниченко Г.А.* Реляционные базы данных: практические приемы оптимальных решений. СПб.: БХВ-Петербург, 2005.

15. *Озкарахан Э.* Машины баз данных и управление базами данных. М.: Мир, 1989.

16. *Ульман Дж.* и др. Системы баз данных. Полный курс. М.: Финансы и статистика, 2003.

17. *Харрингтон Д.* Разработка баз данных / пер. с англ. М.: ДМК Пресс, 2005.

18. *Цикритзис Д., Лоховски Ф.* Модели данных. М.: Финансы и статистика, 1985.

19. *Шекхар Ш, Чаула С.* Основы пространственных баз данных / пер. с англ. М.: КУДИЦ-ОБРАЗ, 2004.

20. ANSI/X3/SPARC Study Group on Data Base Management Systems. Interim Report. FDT Bull. ASM-SIGMOD. 1975. Vol. 7. No. 2.

21. Chen P. P.-S. The Entity-Relationship Model — Toward a Unified View of Data // ACM TODS. 1976. № 1.

22. CODASYL DBTG Report. New York: ACM, 1969.

23. *Codd E.F.* Extending the Database Relational Model to Capture More Meaning. // ACM Trans. Database Syst. 1979. № 4.

24. *Codd E.F., Codd S.B., Salley C.T.* Providing OLAP to User-Analyst: An IT Mandate. E.F. // Codd &Associates. 1993.

25. *Hammer M., Champy J.* Reengineering the Corporation. A Manifesto for Business Revolutions // HarperBusiness. 1993.

26. *Sowa J.F., Zachman J.A.* Extending and Formalizing the Framework for Information System Architecture // IBM System Journal. 1992. Vol. 31. No. 3.

27. *Zachman. John A.* A Framework for Information System Architecture // IBM System Journal. 1987. Vol. 26. No. 3.

Глоссарий

Администратор базы данных (АБД) — лицо или группа лиц, уполномоченных для ведения БД (модификация структуры и содержания БД, активизация доступа пользователей, выполнение других административных функций, которые затрагивают всех пользователей).

Адрес данных — идентификация места расположения данных в памяти.

Адресация — способ размещения и выборки данных в памяти.

Агрегат данных — именованная совокупность элементов данных, представленных простой (линейной) или иерархической (группы или повторяющиеся группы) структурой.

Атрибут — характеристическое свойство объекта, важное с точки зрения предметной области.

База данных (БД) — именованная совокупность взаимосвязанных данных, отображающая состояние объектов и их отношений в некоторой предметной области, используемых несколькими пользователями и хранящихся с минимальной избыточностью.

Банк данных (БнД) — система специально организованных данных, программных, языковых, организационных и технических средств, предназначенных для централизованного накопления и коллективного многоцелевого использования данных.

Документ — агрегат данных в документальных системах, имеющий иерархическую структуру и, кроме форматных полей (элементы или агрегаты данных фиксированной длины), обычно содержащий текстовые поля или символьные последовательности неопределенной длины, логически подразделяющиеся на параграфы, предложения, слова.

Домен — множество возможных значений, из которого берутся значения соответствующего атрибута определенного отношения.

Запись логическая — идентифицируемая (именованная) совокупность элементов или агрегатов данных, воспринимаемая прикладной программой как единое целое при обмене информацией с внешней памятью. С точки зрения структуры запись — это упорядоченная в соответствии с характером взаимосвязей совокупность *полей* (элементов) данных, размещаемых в памяти в соответствии с их *типом*.

Запись физическая — совокупность данных, которая может быть считана или записана как единое целое, обычно одной командой ввода-вывода.

Иерархическая модель данных — использует представление предметной области в форме иерархического дерева, узлы которого связаны по вертикали отношением «предок—потомок».

Инвертированный файл (список) — файл, предназначенный для быстрого произвольного поиска записей по значениям атрибутов, организованный в виде независимых упорядоченных списков адресов элементов хранения (записей, кортежей или документов), соответствующих данному значению атрибута.

Индекс — физическая реализация ключа, обеспечивающая доступ к записям, соответствующим отдельным значениям ключа.

Информационная база — данные, отражающие состояние определенной предметной области и используемые информационной системой. Информационная база состоит из двух компонент: 1) коллекций записей собственно данных и 2) описаний этих данных — *метаданных*. Данные отделены от описаний, но в то же время данные не могут использоваться без обращения к соответствующим описаниям.

Клиент — программа, написанная как пользователем, так и поставщиком СУБД, внешняя или «встроенная» по отношению к СУБД. Программа-клиент организована в виде приложения, работающего «поверх» СУБД и обращающегося для выполнения операций с данными к компонентам СУБД через интерфейс внешнего уровня.

Ключ — значение элемента данных, используемое для идентификации или определения адреса записи.

Ключ внешний — ключ, который в реляционной БД реализует связи «многие к одному» и «один к одному».

Ключ вторичный (альтернативный) — ключ, который идентифицирует некоторую группу записей, имеющих определенное общее свойство. Набор данных может иметь несколько вторичных ключей, необходимость введения которых определяется практической потребностью — оптимизацией процессов нахождения записей по соответствующему ключу.

Ключ первичный (главный) — ключ, значение которого идентифицирует запись единственным образом.

Ключ сцепленный (составной) — несколько элементов данных, которые в совокупности обеспечивают уникальность идентификации каждой записи набора данных.

Кольцевая структура — списковая структура данных, в которой последний элемент указывает на первый, образуя тем самым кольцо.

Контрольная точка — операция согласования состояния базы данных в физических файлах с текущим состоянием системного буфера в операциях обновления БД.

Концепция баз данных — информационная технология интегрированного хранения и обработки данных, в основе которой лежит механизм выделения обрабатывающей программе из всех хранимых данных только тех, которые ей необходимы, и в форме, требуемой именно этой программе.

Метод доступа — метод (способ) организации обмена данными обычно между оперативной и внешней памятью, поддерживаемый ОС, например прямой или последовательный доступ.

Модель данных — модель логического уровня проектирования БД, представляет собой сочетание трех компонентов: (1) структурного компонента, т. е. набора правил, по которым может быть построена БД; (2) управляющего компонента, определяющего типы допустимых операций с данными; (3) поддержки (необязательной) набора ограничений целостности данных, гарантирующей корректность используемых данных.

Модель концептуальная (инфологическая) — семантическая модель предметной области, которая базируется на анализе свойств и природы объектов предметной области и информационных потребностей будущих пользователей разрабатываемой системы.

Модель даталогическая — описание, создаваемое по инфологической модели данных и представленное на языке описания данных конкретной СУБД.

Модель физическая — модель, определяющая размещение и способы поиска данных на внешних запоминающих устройствах.

Моделирования парадигма — условности, определяющие способ *представления* взаимосвязи объектов на уровне *структур данных*. С этой точки зрения различаются реляционные, сетевые, иерархические, объектные, объектно-реляционные, документальные и другие виды моделей.

Независимость данных логическая (физическая) — свойство системы, обеспечивающее возможность изменять логическую (физическую) структуры данных без изменения физической (логической).

Нормализации — декомпозиция реляционной таблицы на две или более с целью ликвидации дублирования данных и потенциальной их противоречивости.

Отношение — структура данных реляционной модели. Имеет две части — заголовок и тело. Заголовок — это множество поименованных атрибутов, каждый из которых задан на определенном домене, а тело — это множество кортежей, содержащих значения атрибутов.

Пользователь БД — программа или человек, обращающийся к базе данных с помощью средств управления данными СУБД.

Предметная область (ПрО) — набор объектов, представляющих интерес для актуальных или предполагаемых пользователей, когда реальный мир отображается совокупностью конкретных и абстрактных понятий, между которыми фиксируются определенные связи.

Представление (View) — это средство создания виртуальной таблицы, содержащей результаты выполнения запроса (оператора SELECT) к одной или нескольким таблицам. Для конечного пользователя представление выглядит как обычная таблица в базе данных, над которой можно выполнять операторы SELECT, INSERT, UPDATE и DELETE. Представления используются для фильтрования и предварительной обработки данных.

Проектирование базы данных — упорядоченный формализованный процесс создания системы взаимосвязанных описаний — таких моделей предметной области, которые связывают (фиксируют) хранимые в базе данные с объектами предметной области, описываемыми этими данными.

Реляционная база данных — база данных, построенная в соответствии с правилами реляционной модели данных.

Сервер — программа, реализующая функции СУБД: определение данных, запись-чтение-удаление данных, поддержку схем внешнего-концептуального-внутреннего уровней, диспетчирование и оптимизацию выполнения запросов, защиту данных.

Сетевая модель данных — предложенное CODASYL расширение иерархической модели, в которой одна запись могла участвовать в нескольких отношениях «предок/потомок». В сетевой модели такие отношения называются *множествами*. Множественные отношения позволяют сетевой базе данных хранить данные, структура которых сложнее обычной иерархии. Множества представлены указателями на физические записи данных. Недостатки — жесткость БД: изменение структуры данных означает перестройку всей базы данных.

Система управления базами данных (СУБД) — совокупность языковых и программных средств, предназначенных для создания, ведения и совместного использования БД многими пользователями.

Словарь данных — каталог всех описаний данных (имен, типов). Может содержать также информацию о пользователях, привилегиях и т. д., доступную только администратору базы данных. Является центральным источником информации для СУБД, АБД и пользователей.

Структура данных — атрибутивная форма представления свойств и связей предметной области, ориентированная на выражение описания данных средствами формальных языков и таким образом учитывающая возможности и ограничения конкретных средств с целью сведения описаний к стандартным типам и регулярным связям. Структура данных с точки зрения программирования — это способ отображения значений в памяти — размер области и порядок ее выделения (который и определит характер процедуры адресации/выборки).

Структура данных линейная — порядок следования элементов данных, который имеет линейный характер и соответствует порядку расположения элементов в памяти.

Структура записей данных — целесообразная (учитывающая особенности физической среды) реализация способов хранения данных и организации доступа к ним как на уровне отдельных записей, так и их элементов.

Структура информации — схематичная форма представления сложных композиционных объектов и связей реальной предметной области, выделяемых как актуально необходимые для решения прикладных задач.

Схема БД концептуальная — абстрагированное описание предметной области с фиксированной (логической) точки зрения.

Схема БД внешняя — представление данных с точки зрения пользователя или прикладной программы

Схема БД внутренняя — физическая структура данных БД.

Таблица — основная структура данных реляционной БД. Состоит из одной или более строк, каждая из которых содержит значения некоторого типа данных (столбцы).

Тип данных — характер данных, определяющий способ представления значения в памяти и соответственно множество стандартных операций (примитивов) преобразования этого значения.

Топология БД — схема распределения компонентов базы данных по физическим носителям, в том числе различным узлам вычислительной сети.

Точка сохранения — место внутри транзакции, определяющее момент сохранения состояния БД.

Транзакция — неделимая с точки зрения воздействия на БД последовательность операторов манипулирования данными (чтения, удаления, вставки, модификации) — такая, что либо результаты всех операторов, входящих в транзакцию, отображаются в БД, либо воздействие всех этих операторов полностью отсутствует.

Уровни представления данных — концептуальный, внутренний и внешний. *Внутренний уровень* — представление БД, которое полностью определяет необходимые условия в первую очередь для организации хранения данных на внешних запоминающих устройствах. *Представление на концептуальном уровне* — обобщенный взгляд на данные с позиций предметной области. *Внешний уровень* — представляет потребности пользователей и прикладных программ.

Файл — именуемая единица информации, поддерживаемая операционной системой, обычно размещаемая на внешнем носителе. Доступ к данным реализуется либо в рамках ОС, либо пользовательскими программами, либо в рамках СУБД, либо комбинированно.

Файл базы данных — физический файл ОС, используемый для размещения БД. Управление данными в таком файле производится совместно ОС и СУБД. Крайние варианты размещения БД по файлам — (1) все данные БД — в одном файле; (2) каждая таблица БД — в отдельном файле ОС. Смешанный вариант размещения, когда база данных состоит из одного или более таблич-

ных пространств, которые в свою очередь состоят из одного или более файлов данных.

Файл логический — файл данных в представлении прикладной задачи, состоящий из логических записей, структура которых может отличаться от структуры физических записей, представляющих информацию в памяти.

Файл физический — файл данных на внешнем носителе, состоящий из физических записей, структура которых может отличаться от структуры логических записей, представляющих информацию в памяти для обработки прикладной программой.

Хранимая процедура (stored procedure) — это набор операторов SQL, которые сервер компилирует в единый план выполнения. Этот план сохраняется в кэше процедур при первом выполнении хранимой процедуры и затем может быть повторно использован уже без рекомпиляции при каждом вызове.

Цепочка данных — организация данных, в которой записи связываются друг с другом посредством указателей.

Элемент данных (элементарное данное) — неделимое именованное данное, характеризующееся типом (например, символьный, числовой, логический, и пр.), длиной (в байтах) и обычно рассчитанное на размещение в одном фрагменте памяти. Это минимальная адресуемая (идентифицируемая) часть памяти — единица данных, на которую можно ссылаться при обращении к данным.

Язык манипулирования данными (ЯМД) — множество инструкций для поддержки основных операций управления данными БД.

Язык определения данных (ЯОД) — средство описания базы данных на структурном уровне.

Язык структурированных запросов (SQL) — язык для реализации всех функциональных возможностей, необходимых для управления БД, в том числе (1) *организации данных*, (2) *обработки данных*, (3) *управления доступом*

Примеры организации данных фактографических и документальных БД

Приложение 1. Физическая структура данных в dBase

Dbase-подобная база данных физически может состоять из специализированных файлов следующего назначения:

- основного файла базы данных;
- memo-файла для хранения длинных полей;
- индексного файла.

Структура основного файла базы данных (тип .DBF)

Файл базы данных состоит из записи заголовка и записей с данными. В записи заголовка, начинающейся с нулевой позиции, определяется структура базы данных.

Количество полей определяет число подзаписей полей. В базе данных для каждого поля существует одна подзапись поля.

Структура заголовка файла данных

Байты	Описание
00	Типы файлов с данными FoxBASE+/dBASE III +, без memo — 0x03 FoxBASE+/dBASE III +, с memo — 0x83 FoxPro/dBASE IV, без memo — 0x03 FoxPro с memo — 0xF5 dBASE IV с memo — 0x8B
01—03	Последнее изменение (ГГММДД)
04—07	Число записей в файле
08—09	Положение первой записи с данными
10—11	Длина одной записи с данными (включая признак удаления)
12—27	Зарезервированы
28	1 — есть составной индексный файл (типа .CDX), 0 — нет
29—31	Зарезервированы
32—*n*	Подзаписи полей (для каждого поля одна подзапись)
N + 1	Признак завершения записи заголовка (0x01)

Структура подзаписи полей

Байты	Описание
00—10	Название поля (максимально — 10 символов)
11	Тип данных: C — символьное; N — числовое; L — логическое; M — типа memo; D — дата; F — с плавающей точкой; P — шаблон
12—15	Расположение поля внутри записи
16	Длина поля (в байтах)
18—32	Зарезервированы

Записи с данными следуют за заголовком и включают в себя фактическое содержимое полей. Длина записи (в байтах) определяется суммированием длин полей, указанных в заголовке.

Записи данных (значений полей) в файле начинаются с позиции, указываемой в записи заголовка в байтах 08—09. Записи начинаются с байта, содержащего признак удаления. Если в этот байт занесен пробел, то запись не удалялась; если же в первом байте звездочка, то запись удалена. За признаком удаления следуют данные из полей, названия которых находятся в подзаписях полей.

Структура memo-файла (тип .FPT)

Файл типа memo содержит одну запись заголовка файла и произвольное число блоков данных.

В записи заголовка располагается указатель на следующий свободный блок и размер блока в байтах, который устанавливается командой SET BLOCKSIZE (или фиксированная длина 512 байтов для файлов типа ***.DBT***) при создании файла. Запись заголовка начинается с нулевой позиции файла и занимает 512 байтов.

За записью заголовка следуют блоки, в которых содержатся заголовок блока и текст memo. В файл базы данных включены номера

блоков, которые используются для ссылки на блоки memo. Расположение блока в memo-файле определяется умножением номера блока на размер блока (находящийся в записи заголовка). Все memo-блоки начинаются с четных адресов границ блоков.

Структура заголовка memo-файла

Байты	Описание
00—03	Расположение следующего свободного блока
04—05	Не используются
06—07	Размер блока (число байтов в блоке)
08—511	Не используются

Заголовок блока memo и текст memo

Байты	Описание
00—03	Сигнатура блока (тип данных в блоке): 0 — шаблон (поле типа шаблон) / 1 — текст (поле типа memo)
04—07	Длина memo (в байтах)
08—*n*	Текст memo (*n* = длина)

Структура индексного файла (тип .IDX)

В индексных файлах располагается одна запись заголовка и одна или больше записей вершин. В записи заголовка находится информация о корневой вершине, текущем размере файла, длине ключа, особенностях индекса и сигнатура, а также представление ключа в коде ASCII, которое можно вывести на печать, и выражения FOR. Запись заголовка начинается с нулевой позиции файла.

Во всех других записях вершин содержатся атрибут, количество существующих ключей и указатели на вершины, располагающиеся слева и справа (на том же уровне) от данной вершины. Помимо этого в них находится группа символов, представляющая значение ключа, и либо указатель на вершину нижнего уровня, либо подлинный номер записи в базе данных. Размер каждой записи равен 512 байтам.

Запись заголовка индексного файла

Байты	Описание
00—03	Указатель на корневую вершину
04—07	Указатель на свободную в списке вершину (–1, если таковая отсутствует)
08—11	Указатель на конец файла (размер файла)
12—13	Длина ключа
14	Особенности индекса (любое из нижеследующих числовых значений либо их сумма): 1 — уникальный индекс; 8 — индекс имеет дополнительный оператор FOR.
15	Сигнатура индекса
16—235	Ключевое выражение (не компилируется; до 220 символов)[1]
236—455	Выражение FOR (не компилируется; до 220 символов, оканчивающееся пустым символом)
456—511	Не используются

Запись вершины индекса

Байты	Описание
00—01	Атрибуты вершины (любое из нижеследующих числовых значений либо их сумма): 0 — вершина индекса; 1 — корневая вершина; 2 — лист
02—03	Количество существующих ключей (0, 1 или больше)
04—07	Указатель на вершину, расположенную непосредственно слева от данной вершины (на том же уровне; –1, если отсутствует)

[1] Тип ключа не запоминается в индексе. Он должен определяться индексным выражением. Если числа используются в качестве ключей, то они подвергаются специальной обработке. Они преобразовываются таким образом, чтобы их можно было отсортировать с помощью такой же схемы упорядочения в коде ASCII, что и символы, т. е. преобразовать число в формат с плавающей точкой IEEE и изменить на противоположный порядок следования байтов с порядка Intel на порядок слева направо. Если число отрицательное, взять логическое дополнение числа (изменить на противоположные все 64 бита, 1 на 0 и 0 на 1), иначе инвертировать только самый левый бит.

Окончание табл.

Байты	Описание
08—11	Указатель на вершину, расположенную непосредственно справа от данной вершины (на том же уровне; −1, если отсутствует)
12—511	До 500 символов, включающих в себя значение ключа для длины ключа с четырехбайтовым шестнадцатиричным числом (хранящемся в обычном формате слева направо). Если вершина является листом (атрибут = 02 или 03), тогда четыре байта содержат подлинный номер, номер в базе данных в шестнадцатиричном формате — иначе 4 байта содержат внутриндексный указатель[1]

Структура компактного индексного файла (тип .IDX)

Структура записи заголовка компактного индексного файла

Байты	Описание
00—03	Указатель на корневую вершину
04—07	Указатель на свободную в списке вершину (−1, если таковая отсутствует)
08—11	Резервируются для внутреннего использования
12—13	Длина ключа
14	Особенности индекса (любое из нижеследующих значений либо их сумма): 1 — уникальный индекс; 8 — индекс имеет дополнительный оператор FOR; 32 — формат компактного индекса; 64 — заголовок составного индекса
15	Сигнатура индекса
16—35	Зарезервированы для внутреннего использования
36—501	Зарезервированы для внутреннего использования
502—503	По возрастанию или убыванию: 0 — возрастание; 1 — убывание

[1] В вершине-листе все, что отлично от символьных строк, числа, используемые в качестве значений ключей, и четырехбайтовые номера представляются в байтах, порядок которых изменен на противоположный (в формате Intel 8086).

Окончание табл.

Байты	Описание
504—505	Зарезервированы для внутреннего использования
506—507	Длина пула выражения FOR
508—509	Зарезервированы для внутреннего использования
510—511	Длина пула выражения FOR
510—1023	Пул выражения ключа (не компилируется)

Структура записи внутренней вершины для компактного индекса

00—01	Атрибуты вершины (любое из нижеследующих числовых значений либо их сумма): 0 — индексная вершина; 1 — корневая вершина; 2 — вершина-лист
00—01	Атрибуты вершины (любое из нижеследующих числовых значений либо их сумма): 0 — индексная вершина; 1 — корневая вершина; 2 — вершина-лист
02—03	Число существующих ключей (0, 1 или больше)
04—07	Указатель на вершину, расположенную непосредственно слева от данной вершины (на том же уровне; –1 — если отсутствует)
08—11	Указатель на вершину, расположенную непосредственно справа от данной вершины (на том же уровне; –1 — если отсутствует)
12—511	До 500 символов, включающих в себя значение ключа для длины ключа с четырехбайтовым шестнадцатиричным числом (хранящемся в обычном формате слева направо). Эта вершина всегда содержит ключ индекса, номер записи и внутриндексный указатель[1]. Комбинация из значения ключа и четырехбайтового числа будет повторена столько раз, сколько задано в байтах 02—03

[1] Каждый элемент состоит из номера записи, запасного байтового счетчика и хвостового байтового счетчика, все в сжатом виде. Текст ключа помещается в логический конец вершины, обрабатывается он в обратном направлении, что позволяет находить элементы предшествующих ключей.

Структура записи внешней вершины для компактного индекса

Байты	Описание
00—01	Атрибуты вершины (любое из нижеследующих числовых значений либо их сумма): 0 — индексная вершина; 1 — корневая вершина; 2 — вершина-лист
02—03	Число существующих ключей (0, 1 или больше)
04—07	Указатель на вершину, расположенную непосредственно слева от данной вершины (на том же уровне; –1 — если отсутствует)
08—11	Указатель на вершину, расположенную непосредственно справа от данной вершины (на том же уровне; –1 — если отсутствует)
12—13	Свободное для распределения пространство в вершине
14—17	Маска номера записи
18	Маска запасного байтового счетчика
19	Маска хвостового байтового счетчика
20	Количество битов, используемых для номера записи
21	Количество битов, используемых для запасного счетчика
22	Количество битов, используемых для хвостового счетчика
23	Количество байтов, содержащих номер записи, запасной счетчик и хвостовой счетчик
24—511	Ключи индексов и информация

Приложение 2. Физическая структура данных в MS SQL Server

Файлы операционной системы в MS SQL Server представляются как *нумерованные устройства* для хранения БД. Каждое устройство разбивается на виртуальные страницы по 8 Кбайт.

MS SQL Server используется следующая иерархия понятий.

База данных — некоторый объем файлового физического пространства для размещения данных, принадлежащих одной логической базе.

Файлы БД. Каждая база данных состоит не менее чем из двух файлов. Один из них отводится под журнал транзакций. Отдельный файл данных может принадлежать только одной базе данных.

Экстент. Пространство для хранения объектов выделяется блоками (*экстентами*) по 8 следующих друг за другом страниц размером 8К. Экстент является единицей выделения пространства. Поэтому при создании БД нужно указывать размер файла с точностью до 64 Кбайт.

Страница. Файлы делятся на страницы размером по 8 Кбайт каждая. Логический номер страницы складывается из внутреннего номера базы данных, номера файла и номера страницы в файле. В рамках БД файлы нумеруются, начиная с 1, и так же нумеруются страницы в рамках файла.

Используется два типа экстентов: *однородные* и *смешанные*. Однородные экстенты всегда принадлежат только одному объекту. Смешанный экстент может использоваться несколькими объектами.

В SQL Server существуют несколько типов страниц.

Следующие типы страниц относятся к хранению и поиску информации:

- страниц данных;
- индексных страниц;
- текстовых страниц;
- страницы журнала транзакций;

Кроме этого используются также страницы размещения:

- карты распределения блоков (основная и вторичная);
- карты свободного пространства;
- индексные карты размещения.

На странице всегда (в отличие от экстента) хранится однородная информация. Все страницы имеют *заголовок*, в котором хранится общая информация, используемая ядром СУБД для работы со страницами:

- номер страницы в формате <*номер файла, номер страницы*>;
- идентификатор объекта, которому принадлежит страница;
- индекс и уровень внутри индексного дерева, которому принадлежит страница;
- количество строк на странице;
- общий объем свободного пространства на странице;
- указатель на свободное пространство после последней строки на странице;
- минимальная длина строки на странице;
- объем зарезервированного пространства.

После заголовка следует информация о статусе страницы в картах распределения блоков и карте свободного пространства.

Страницы размещения

SQL Server использует три типа страниц размещения: карты распределения экстентов, карты свободного пространства, индексные карты размещения.

Карты распределения экстентов

Карта распределения экстентов состоит из стандартного заголовка и битового массива в 64 000 битов. Каждый бит характеризует один экстент. Поэтому одна страница карты распределения описывает пространство в 64 000 экстентов или 4 Гбайта данных.

При отведении пространства используются два типа карт распределения экстентов:

- глобальная карта распределения (Global Allocation Map — GAM) хранит информацию об использовании экстентов. Если бит установлен в 0, то экстент занят данными, если в 1 — то экстент свободен;
- вторичная глобальная карта распределения (Secondary Global Allocation Map — SGAM) хранит информацию о типе экстентов. Если бит установлен в 1, то соответствующий экстент смешанный и минимум одна страница в нем свободна, в остальных случаях бит равен 0.

Карты свободного пространства

Карта свободного пространства (Page Free Space Page — PFS) отражает степень заполнения страниц. Каждая PFS-страница хранит информацию о 8000 страницах — по одному байту на страницу. Каждый байт представляет собой битовую карту, которая сообщает о степени занятости страницы и о том, принадлежит ли она объекту.

Карта распределения размещается с первой страницы файла БД. Страницы повторяются через каждые 8000 страниц.

Первая страница PFS после стандартного *заголовка страницы* содержит *заголовок файла* (его описание), затем размещается сам блок PFS. Вторая страница — это GAM, третья — SGAM. Карты распределения экстентов повторяются через каждые 512 000 страниц.

Карты размещения

Для организации связи между экстентами и расположенными на них объектами используются *индексные карты размещения* (Index

Allocation Map — IAM). Каждая таблица или индекс имеют одну или более страниц IAM. В каждом файле, в котором размещаются таблица или индекс, существует минимум одна карта размещения для этой таблицы или индекса. Страницы IAM размещаются произвольно внутри файла и отводятся по мере необходимости. IAM объединены друг с другом в цепочку двунаправленными ссылками. Указатель на первую карту размещения содержится в поле FirstIAM системной таблицы Sysindex.

Каждая IAM описывает некоторый диапазон экстентов и представляет собой битовую карту: если бит установлен в 1, то в данном экстенте есть страницы, принадлежащие данному объекту, если в 0 — то нет.

Все страницы размещения не связаны напрямую с некоторым объектом БД, они соответствуют некоторой системной информации, поэтому параметр «идентификатор объекта» для всех этих страниц одинаков и равен 99.

Страницы данных

Страницы данных используются для хранения собственно данных. Структурно страницу данных можно подразделить на три зоны: заголовок, строки данных и таблицу размещения строк (слоты). Связь между страницами и объектами реализует специальная структура — карты размещения.

Строка данных должна полностью умещаться на странице, поэтому существуют ограничения на длину строки. Размер страницы 8 Кбайт, 96 байт занимает заголовок. Кроме того, в таблице размещения каждому слоту отводится по 4 байта для каждой строки, размещенной на странице.

Строки данных на странице не обязательно хранятся непрерывно. При удалении строки пустое пространство помечается как свободное, и потом его может занять новая строка, перемещения строк не происходит. Адрес (смещение) на странице и длина строки фиксируются в *слоте* (Slot).

Если таблица не имеет кластеризованного индекса, то номер слота является идентификатором строки, пока не будет удалена соответствующая строка. Если же таблица имеет кластеризованный индекс, то слоты располагаются в порядке, задаваемом индексом.

Первые страницы данных таблиц БД расположены не подряд: сразу за первой страницей данных таблицы следует ее индексная карта размещения.

Для более эффективного управления дисковым пространством SQL Server не выделяет создаваемым таблицам сразу целый экстент. Для новой таблицы или индекса, как правило, выделяется место на смешанном экстенте. Когда объем таблицы или индекса увеличивается до восьми страниц, все последующие выделяемые экстенты будут однородными. Соответственно, если на смешанных экстентах места нет, а объем таблицы не достиг еще восьми страниц, то выделяемый новый экстент будет объявлен смешанным. Например, если таблица занимает две страницы на смешанном экстенте и в нее еще добавляется сразу шесть записей, то при отсутствии свободных страниц на смешанных экстентах будет выделен новый смешанный экстент, и на нем разместятся шесть записей. Потом, если добавляется еще одна запись, будет выделен полный новый однородный экстент, и на нем размещена эта новая запись.

Строки данных

Данные хранятся на страницах в виде строк. Каждая строка кроме собственно данных хранит дополнительную форматирующую информацию. Длина строки зависит от типов полей таблицы. Независимо от объявления каждая строка имеет поле с количеством полей переменной длины (к ним относятся также поля фиксированной длины, допускающие неопределенные значения NULL, которые при этом резервируют пространство, указанное в определении поля).

Фиксированные поля вместе с описателями хранятся до полей переменной длины. Поля фиксированной длины всегда занимают свою полную длину, значение NULL задается специальным флагом.

В каждой строке хранится общая длина строки и текущие длины полей переменной длины. Данные считываются последовательно с начального адреса.

Вторая часть — это необязательная область, она существует только тогда, когда в записи имеются поля переменной длины, и включает:

- указатель на местоположение полей переменной длины;
- собственно значения полей переменной длины.

Текстовые страницы

Текстовая страница может содержать несколько текстовых полей.

Строка данных содержит указатель на корневую структуру. Собственно данные хранятся в виде сбалансированного B-дерева.

Данные длиной менее 64 байт хранятся в корневой структуре.

Для данных до 32 Кбайт корневая структура может адресовать 4 блока данных (это не экстенты страниц) до 8 Кбайт каждый. Блоки наращиваются до 8 Кбайт (реально на одной текстовой странице может быть размещено до 8080 байт).

Если же длина текстового поля более 32 Кбайт, то строятся промежуточные узлы.

Индексы

Кластеризованный индекс представляет собой двоичное дерево, в котором на нулевом уровне (уровне листьев) содержатся страницы актуальных данных таблицы, а физическое расположение информации в данном индексе логически упорядочено.

В случае некластеризованных индексов страницы уровня листа содержат не актуальные данные таблицы (как в случае кластеризованного индекса), а указатель на строку данных, включающий номер страницы данных и порядковый номер записи на станице. Некластеризованный индекс не требует физического переупорядочения строк данных таблицы.

Двоичные деревья являются *динамически поддерживаемыми структурами*, т.е при вставке, удалении или обновлении строк данных информация в индексах также должна быть изменена для отражения выполненных в таблице изменений. Для обработки страниц индексов требуются дополнительные операции ввода-вывода.

Но при разбивке[1] страницы кластеризованного индекса не требуется вносить изменения в информацию некластеризованных индексов, так как в SQL Server 7 эти дополнительные операции ввода-вы-

[1] При добавлении в кластеризованный индекс данные вставляются в таблицу в правильной физической последовательности, соответствующей логической упорядоченности индекса. Поэтому может потребоваться сместить остальные строки таблицы вверх или вниз, в зависимости от места вставки данных. Все это связано с выполнением дополнительных операций ввода-вывода, необходимых для обработки данных индекса. Со временем, по мере накопления данных на странице, при вставке очередной записи может потребоваться разбить информацию страницы на две части, так как для новой записи уже не будет хватать свободного места. В результате часть записей данных будет перенесена на новую страницу. В предыдущих версиях SQL Server эта ситуация сопровождалась обновлением данных во всех некластеризованных индексах таблицы с целью отражения перемещения части строк на новую страницу.

вода полностью исключены за счет использования в некластеризованных индексах значений ключей кластеризованного индекса вместо номеров физических страниц, в случае когда таблица имеет оба типа индексов.

Индексы таблиц хранятся в виде страниц. Каждая страница размером 8192 байта включает заголовок, имеющий длину 96 байт. Еще один фрагмент страницы используется для размещения других структур данных, например информации о переполнении строк. Вся оставшаяся часть страницы (8060 байт) предназначена для размещения данных. Каждая строка включает элемент индекса (значение индексируемого поля таблицы) и идентификатор RowID (включающий идентификатор файла, номер страницы, номер строки), указывающий на соответствующую запись в таблице.

Организация и оптимизация доступа к данным

Вследствие объективно существующей разницы в скорости работы процессоров и оперативной памяти, с одной стороны, и устройств внешней памяти, с другой — буферизация страниц базы данных в оперативной памяти — единственно реальный способ достижения удовлетворительной эффективности СУБД. Кроме этого используется механизм *распределенного хранения информации* — расщепления данных между файлами и файловыми группами, физически размещаемыми на разных устройствах или RAID-массивах. Логически такое устройство представляется как единое целое, но на самом деле состоит из нескольких физических дисков. Данные на дисках размещаются блоками одной длины и, таким образом, легко могут быть распределены по всем дискам.

Стратегия буферизации, применяемая в операционных средах, не соответствует целям и задачам СУБД, поэтому для оптимизации обработки данных одной из главных задач СУБД является создание эффективной системы управления процессом буферизации.

Память, управляемая СУБД, состоит из нескольких типов буферов:

- буфера страниц данных, с которыми работает СУБД;
- буфера страниц журнала транзакций, которые отражают процесс выполнения транзакции — последовательности операций над БД, переводящей БД из одного непротиворечивого состояния в другое непротиворечивое состояние;

- системные буферы, которые содержат общую информацию о БД, о пользователях, о физической структуре БД, о базе метаданных.

Если бы запись об изменении базы данных реально немедленно записывалась во внешнюю память, это привело бы к существенному замедлению работы системы. Поэтому записи в журнал тоже буферизуются: при нормальной работе очередная страница выталкивается во внешнюю память журнала только при полном наполнении записями.

Но поскольку имеются два вида буферов, содержащих взаимосвязанную информацию, — буфер журнала и буфер страниц оперативной памяти, которые могут выталкиваться во внешнюю память, буферы выделяются не для каждого пользовательского процесса, а для всех процессов сервера. Это позволяет увеличить степень параллелизма при исполнении клиентских процессов.

Приложение 3. Физическая структура данных в СУБД Oracle

Единицами распределения пространства базы данных в СУБД Oracle являются блоки данных, экстенты и сегменты.

При создании объекта БД, такого как таблица или индекс, создается сегмент этого объекта. Место для сегмента данных распределяется в одном или нескольких файлах данных, составляющих *табличное пространство* (рис. П3.1).

Сегмент объекта может размещаться лишь в одном табличном пространстве базы данных. Экстенты одного сегмента распределяют-

Рис. П3.1. Табличное пространство Oracle

ся *блоками данных,* возможно, в нескольких файлах табличного пространства: таким образом, объект может «занимать» один или несколько файлов данных. Но при этом отдельный экстент не может находиться в нескольких файлах (рис. П3.2).

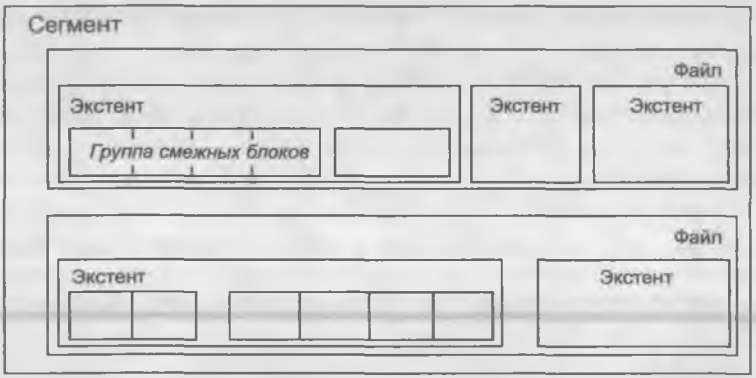


Рис. П3.2. Структура сегмента

Общий объем памяти табличного пространства определяется суммой размеров файлов данных, составляющих это табличное пространство. Суммарный объем всех табличных пространств базы данных составляет общий объем базы данных.

Сегменты

Сегмент — это совокупность экстентов, распределенных для специфического типа данных и находящихся в одном и том же табличном пространстве. База данных Oracle может содержать четыре различных типа сегментов: сегмент данных, сегмент индекса, сегмент отката, временный сегмент.

Каждый тип сегмента состоит из одного или нескольких экстентов. Когда существующее пространство в сегменте полностью использовано, выделяется новый сегмент. Поскольку экстенты распределяются по мере поступления данных, они не обязательно будут смежными на диске и могут быть распределены между различными файлами. Отдельный экстент, однако, не может размещаться в нескольких файлах.

Каждый сегмент в базе данных содержит заголовок, который описывает характеристики этого сегмента и оглавление (список) экстентов в этом сегменте.

Сегменты данных. Каждая некластеризованная таблица или кластер в БД Oracle имеет единственный сегмент данных, содержащий все данные этого объекта.

Сегменты индекса. Каждый индекс в базе данных Oracle имеет единственный сегмент индекса, содержащий все данные этого индекса.

Сегмент данных таблицы и сегмент индекса не обязательно размещаются в одном и том же табличном пространстве.

Сегменты отката. Каждая база данных содержит один или несколько сегментов отката. Сегмент отката — это часть базы данных, в которой записываются действия транзакций, которые, возможно, придется отменить.

Временные сегменты. При обработке запросов Oracle часто требует временного рабочего пространства для промежуточных этапов обработки предложения SQL. Это дисковое пространство, называемое ВРЕМЕННЫМ СЕГМЕНТОМ, распределяется автоматически.

Экстенты

Экстент — это смежные блоки данных, выделенные для хранения специфического типа информации. При создании объекта, такого как таблица или индекс, создается сегмент этого объекта в предназначенном табличном пространстве.

При распределении нового экстента ORACLE отыскивает в табличном пространстве, содержащем сегмент, первый свободный непрерывный участок блоков данных, размер которого больше запрашиваемого размера по крайней мере на один блок. Если запрашиваемый участок непрерывных блоков не найден, ORACLE соединяет все свободные блоки данных в соответствующем табличном пространстве, так что формируются группы непрерывных блоков большего размера, при этом фоновый процесс периодически объединяет смежное свободное пространство.

Блоки данных

Данные БД хранятся в *блоках данных,* называемых также логическими блоками или страницами. Один блок данных занимает пространство фиксированного размера (байт) физического пространства на диске, который устанавливается для каждой базы данных при ее создании. Этот размер кратен размеру блока операционной системы, но не превышает определенный максимум. Блок данных — наименьшая единица ввода-вывода СУБД.

Формат блока данных, независимо от того, содержит ли он данные таблицы, индекса или кластера, включает: *Общий и переменный заголовок, Оглавление таблиц, Оглавление строк, Свободное пространство, Строки данных.*

Заголовок содержит общую информацию блока, такую как адрес блока и тип сегмента.

Оглавление таблиц содержит информацию о том, строки каких таблиц размещены в этом блоке.

Оглавление строк содержит информацию о действительных строках в блоке (включая адреса каждой порции строки в области данных строк). После того, как в оглавлении строк распределено пространство, это пространство не освобождается при удалении строки. Это пространство будет использовано повторно лишь тогда, когда в блок вставляются новые строки.

Строки данных содержат данные таблицы или индекса. Строки могут переходить из блока в блок, если все данные строки не умещаются в один блок. В таком случае данные этой строки сохраняются в виде *цепочки блоков* данных, резервируемых в этом сегменте. Если строка в блоке данных обновляется так, что общая длина строки увеличивается, а свободное пространство в блоке заполнено, то данные всей строки мигрируют, при условии что в новом блоке поместится вся строка. На месте мигрировавшей строки записывается указатель на новый блок, содержащий мигрировавшую строку.

Свободное пространство в блоке используется для вставки новых строк и для обновлений строк, требующих дополнительного пространства (например, при замене пустых хвостовых значений на непустые значения). Будут ли конкретные вставки действительно осуществляться в данном блоке — зависит от значения параметра управления пространством и от текущей величины свободного пространства в блоке.

Типы индексов

В СУБД Oracle используются следующие типы индексов:

- индексы на основе B-дерева;
- индексы с реверсированным ключом;
- индексы по убыванию. Индексы по убыванию позволяют отсортировать данные в структуре индекса от больших значений к меньшим;
- индексы на основе битовых карт;

- функциональные индексы;
- полнотекстовые индексы.

Существует возможность создания кластерных индексов.

Кластеры

Кластеры — это необязательный способ хранения данных таблиц. Кластер представляет собой группу таблиц, разделяющих одни и те же блоки данных, потому что они имеют общие столбцы и часто используются вместе.

Так как кластеры хранят связанные строки разных таблиц вместе в одних и тех же блоках данных, при правильном использовании кластеров достигаются два главных преимущества:

- сокращается дисковый ввод-вывод и улучшается время доступа для соединений по кластеризованным таблицам.

В кластере *значение ключа кластера* представляет собой совокупность значений столбцов ключа кластера для конкретной строки. Каждое значение ключа кластера хранится лишь один раз, как в кластере, так и в индексе кластера, независимо от того, сколько строк в различных таблицах содержат это значение. Поэтому для хранения данных и индекса связанных таблиц требуется меньше места, чем для формата некластеризованных таблиц.

Приложение 4. Документальная информационно-поисковая система

Организация данных и механизмы поиска в базах данных документальных информационных систем построены на тех же принципах, что и фактографические системы. Однако в физической реализации есть и существенные отличия, которые обусловлены в первую очередь информационной природой элементов данных:

1. Запись базы данных — документ, который задается как набор в общем случае *необязательных* полей, для каждого из которых определены имя и тип. Допустимы большинство стандартных типов (так называемые «форматные» поля, задающие числовые, символьные и другие величины), а также текстовые. Текстовые поля имеют переменную длину и композиционную структуру, не имеющую прямых аналогов среди стандартных типов языков программирования: текстовое поле состоит из параграфов; параграф — из предложений;

предложение — из слов. При этом идентифицируемым (адресуемым атомарным) элементом данных с точки зрения хранения будет *поле*, а с точки зрения поиска (атомарным семантически значимым) — *слово*. Вследствие этого поисковые структуры строятся в виде инвертированных файлов.

2. Семантическая природа текстовых полей, представляющих смысл в основном на естественном языке, определяет необходимость учитывать важнейшие свойства используемых терминов: синонимию, полисемию, омонимию, контекстную обусловленность смысла отдельного слова и возможность выразить один смысл многими способами. Вследствие этого поисковые *индексы* могут быть отличны от соответствующих словоформ поля.

На рис. П4.1 приведена принципиальная схема организации данных для представления и поиска информации диалоговой системы поиска документов STAIRS (Storage and Information Retrieval System), разработанной фирмой IBM в 1970-х гг. Данная структура характерна и для большинства современных АИПС.

Физическая структура БД рассматриваемой системы включает в себя четыре файла операционной системы:

- файл частотного словаря, устанавливающий соответствие между словом, встречающимся в БД, его кодом и частотой, используется при текстовом поиске;

Рис. П4.1. Организация данных в диалоговой системы поиска документов STAIRS

- инверсный (инвертированный, обратный) список, содержащий для каждого слова БД список документов, его содержащих, используется при текстовом поиске;
- текстовый файл, содержащий собственно документы, используется при выдаче (просмотре) документов;
- прямой, последовательный файл, содержащий «собранные» в одну строку фиксированной длины форматные поля и список двухбайтовых кодов слов, находящихся в тексте данного документа. При необходимости в соответствующих местах находятся разделители сегментов и/или предложений. Файл используется при форматном поиске и при наличии в запросах конструкций *SENT, SEGM, CTX*.

На рис. П4.2 детально представлен *словарь слов,* в котором содержится перечень слов, встречающихся в документах. Ввиду значительных размеров словаря его организация должна предусматривать наличие специального индекса, представленного *матрицей пар знаков*. Каждой паре знаков поставлен в соответствие указатель на *блок словаря*, содержащий группу слов, начинающихся с этих знаков. Знаками могут быть буквы, цифры, а также специальные символы. Второй знак может быть пробелом. Группы слов в словаре имеют переменную длину. Первые два знака слов, содержащихся в словаре, отсутствуют, но они показаны на рисунке, чтобы облегчить понимание структуры файла. Некоторые слова в словаре могут иметь одинаковый смысл; такие слова связаны с помощью специального указателя «синоним» (на рисунке связи данного типа показаны штриховыми стрелками).

Каждому слову поставлен в соответствие указатель на *списки экземпляров,* являющихся перечнем документов, в которых встречается данное слово. Каждый список экземпляров содержит заголовок, из которого можно узнать число экземпляров слова во всем файле документов, а также число документов, в которых это слово встречается.

Система присваивает каждому документу уникальный номер. Этот номер является внутрисистемным и не связан с номерами, по которым пользователь может получить данный документ где-нибудь вне системы. В списке экземпляров, соответствующем какому-либо слову, содержатся внутрисистемные номера всех документов, в которых оно встречается. Поисковый критерий может включать требование *поиска всех документов, содержащих одновременно два специфических слова*. Например, можно осуществлять поиск документа, в котором содержится как слово ORANGUTANG, так и слово OSTRICH. В этом случае система находит множество документов, содержащих

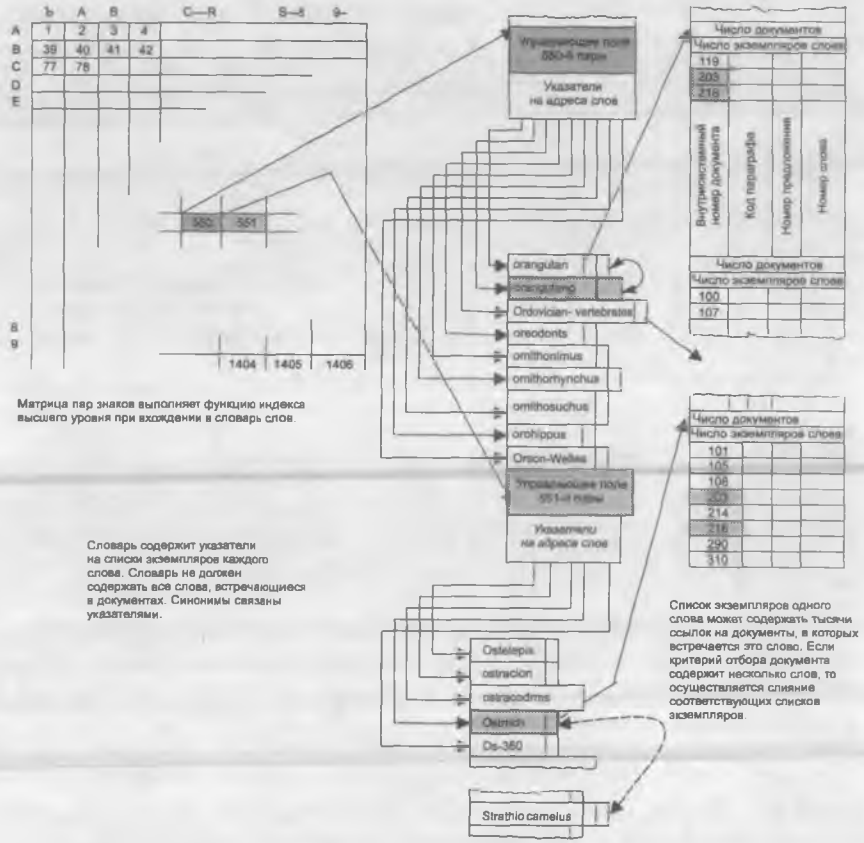


Рис. П4.2. Организация поисковых индексов АИПС STAIRS

первое слово, а затем множество документов, содержащих второе слово, и путем их пересечения определяет множество документов, содержащих как первое, так и второе слово.

На рис. П4.3 показан *файл документов*, каждому из которых система сама присваивает внутренний порядковый номер. Документы состоят из параграфов и текстов, причем тексты также пронумерованы. Каждому параграфу присвоен специальный код, определяющий его тип (например, заголовок, автор, аннотация и т. д.).

Внутрисистемный номер документа является ключом к *индексу документов*. Этот индекс содержит адреса соответствующих документов в памяти. В принципе *можно* хранить эти адресные указатели не-

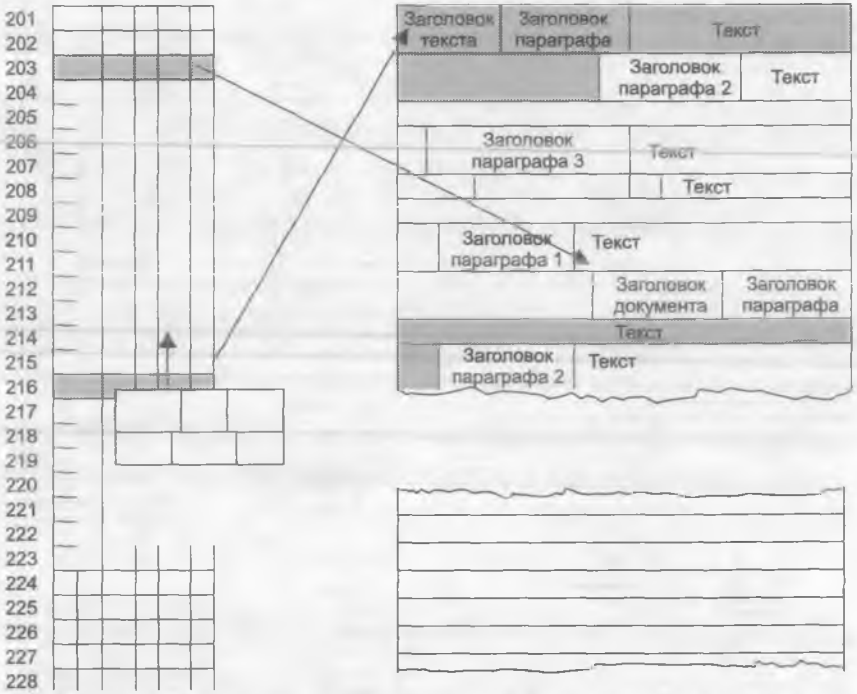


Рис. П4.3. Организация поисковых файлов документов АИПС STAIRS

посредственно в списке экземпляров, но это нецелесообразно, так как объем памяти, необходимый для хранения адреса, больше объема памяти, необходимого для хранения номера документа. Индекс документов содержит не только адреса, но также некоторые вспомогательные сведения о документах. К этим сведениям относятся внешний номер документа, признак удаления документа, указывающий, какие параграфы документа (или документ в целом) исключены из файла, а также уровень секретности.

В состав документов могут входить *параграфы различных типов*, поэтому пользователь может потребовать, чтобы заданное слово содержалось в названии документа, аннотации, введении или каком-либо конкретном параграфе. В критерии отбора можно указывать автора, место издания документа и дату издания. Независимо от содержания критерия отбора поиск документа осуществляется на уровне списка экземпляров без необходимости входа в файл документов.

Оглавление

Голицына Ольга Леонидовна
Максимов Николай Вениаминович
Попов Игорь Иванович

Базы данных

Учебное издание

Редактор *А.В. Волковицкая*
Корректор *Т.И. Якушкина*
Компьютерная верстка *И.В. Кондратьевой*
Оформление серии *П. Родькина*

Подписано в печать 02.02.2012. Формат 60×90/16.
Гарнитура «Таймс». Усл. печ. л. 25,0. Уч.-изд. л. 25,6.
Печать офсетная. Бумага офсетная. Тираж 1000 экз.
Заказ № 3149.

Издательство «ФОРУМ»
101990, Москва — Центр, Колпачный пер., д. 9а
Тел./факс: (495) 625-32-07, 625-52-43
E-mail: forum-knigi@mail.ru

Отдел продаж издательства «ФОРУМ»:

101990, Москва — Центр, Колпачный пер., д. 9а
Тел./факс: (495) 625-52-43
E-mail: forum-ir@mail.ru
www.forum-books.ru

Книги издательства «ФОРУМ»
вы также можете приобрести:

Отдел продаж «ИНФРА-М»
127282, Москва, ул. Полярная, д. 31в
Тел.: (495) 380-05-40 (доб. 252)
Факс: (495) 363-92-12

Отдел «Книга-почтой»
E-mail: podpiska@infra-m.ru;
books@infra-m.ru

Отпечатано с электронных носителей издательства.
ОАО "Тверской полиграфический комбинат". 170024, г. Тверь, пр-т Ленина, 5.
Телефон: (4822) 44-52-03, 44-50-34, Телефон/факс: (4822)44-42-15
Home page - www.tverpk.ru Электронная почта (E-mail) - sales@tverpk.ru