Глава 1

Мир баз данных

Современные технологии баз данных являются одним из определяющих факторов успеха в любой отрасли бизнеса, обеспечивая хранение корпоративной информации, представление данных для пользователей и клиентов в среде World Wide Web и поддержку многих других процессов. Помимо того, базы данных составляют основу разнообразных научных проектов. Они позволяют накапливать информацию, собранную астрономами, исследователями генотипа человека, биохимиками, изучающими свойства протеинов, и специалистами многих других отраслей знания.

Мощь баз данных зиждется на результатах исследований и технологических разработок, полученных на протяжении нескольких последних десятилетий, и заключена в специализированных программных продуктах, которые принято называть *системами управления базами данных* (СУБД) (Database Management Systems — DBMS), или просто *системами баз данных* (database systems). СУБД — это эффективный инструмент сбора больших порций информации и действенного управления ими, позволяющий сохранять данные в целости и безопасности на протяжении длительного времени. СУБД относятся к категории наиболее сложных программных продуктов, имеющихся на рынке в настоящее время. СУБД предлагают пользователям функциональные возможности, перечисленные ниже.

1. *Средства постоянного хранения данных.* СУБД, подобно файловым системам, поддерживают возможности хранения чрезвычайно больших фрагментов данных, которые существуют независимо от каких бы то ни было процессов их использования. СУБД, однако, значительно превосходят файловые системы в отношении гибкости представления информации, предлагая структуры, обеспечивающие эффективный доступ к большим порциям данных.
2. *Интерфейс программирования*. СУБД позволяет пользователю или прикладной программе обращаться к данным и изменять их посредством команд развитого языка запросов. Преимущества СУБД в сравнении с файловыми системами проявляются и в том, что первые дают возможность манипулировать данными самыми разнообразными способами, гораздо более гибкими, нежели обычные операции чтения и записи файлов.
3. *Управление транзакциями*. СУБД поддерживают параллельный доступ к данным, т.е. возможность единовременного обращения к одной и той же порции данных со стороны нескольких различных процессов, называемых *транзакциями* (transactions). Чтобы избежать некоторых нежелательных последствий подобного обращения, СУБД реали-

зуют механизмы обеспечения *изолированности* (isolation) транзакций (транзакции выполняются независимо одна от другой, как если бы они активизировались строго последовательно), их *атомарности* (atomicity) (каждая транзакция либо выполняется целиком, либо не выполняется вовсе) и *устойчивости* (durability) (системы содержат средства надежного сохранения результатов выполнения транзакций и самовосстановления после различного рода ошибок и сбоев) .

1.1. Эволюция систем баз данных

Что такое база данных? По существу, это не что иное, как набор порций информации, существующий в течение длительного периода времени (возможно, исчисляемого годами или даже десятилетиями). Термином *база данных* (database) в соответствии с принятой традицией обозначают набор данных, находящийся под контролем СУБД. Добротная СУБД обязана обеспечить реализацию следующих требований.

1. Позволять пользователям создавать новые базы данных и определять их *схемы* (schemata) (логические структуры данных) с помощью некоторого специализированного языка, называемого *языком определения данных* (Data Definition Language — DDL).
2. Предлагать пользователям возможности задания *запросов* (queries) (слово "запрос", в данном случае обладающее известным жаргонным оттенком, означает вопрос, затрагивающий те или иные аспекты информации, хранящейся в базе данных) и модификации данных средствами соответствующего *языка запросов* (query language), или *языка управления данными* (Data Manipulation Language — DML).
3. Поддерживать способность сохранения больших объемов информации — до многих гигабайтов и более — на протяжении длительных периодов времени, предотвращая опасность несанкционированного доступа к данным и гарантируя эффективность операций их просмотра и изменения.
4. Управлять единовременным доступом к данным со стороны многих пользователей, исключая возможность влияния действий одного пользователя на результаты, получаемые другим, и запрещая совместное обращение к данным, чреватое их порчей.

1.1.1. Первые СУБД

Появление первых коммерческих систем управления базами данных датируется концом 1960-х годов. Непосредственными предшественниками таких систем были файловые системы, удовлетворявшие только некоторым из требований, перечисленных выше (см. п. 3). Файловые системы действительно пригодны для хранения обширных фрагментов данных в течение длительного времени, но они не способны гарантировать, что данные, не подвергшиеся резервному копированию, не будут испорчены или утеряны, и не поддерживают эффективные инструменты доступа к элементам данных, положение которых в определенном файле заранее не известно.

Файловые системы не отвечают и условиям п. 2, т.е. не предлагают языка запросов, подходящего для обращения к данным внутри файла. Обеспечиваемая ими реализация требований п. 1, предполагающих наличие возможности создания схем данных, ограничена способностью определения структур каталогов. Наконец, файловые системы совершенно не удовлетворяют условиям п. 4. Даже если система и не препятствует параллельному доступу к файлу со стороны нескольких пользователей или процессов, она не в состоянии предотвратить возникновение ситуаций, когда, например, два пользователя в один и тот же момент времени пытаются внести изменения в один файл: результаты действий одного из них наверняка будут утрачены.

К числу первых серьезных программных приложений СУБД относились те, в которых предполагалось, что данные состоят из большого числа элементов малого объема; для обработки таких элементов требовалось выполнять множество элементарных запросов и операций модификации. Ниже кратко рассмотрены некоторые из ранних приложений баз данных.

Системы бронирования авиабилетов

Система подобного типа имеет дело со следующими элементами данных:

1) сведениями о резервировании конкретным пассажиром места на определенный авиарейс, включая информацию о номере места и обеденном меню;
2) информацией о полетах — аэропортах отправления и назначения для каждого рейса, сроках отправки и прибытия, принадлежности воздушных судов тем или иным компаниям, экипажах и т.п.;
3) данными о ценах на авиабилеты, поступивших заявках и наличии свободных мест.

В типичных запросах требуется выяснить, какие рейсы из одного заданного аэропорта в другой близки по времени отправления к указанному календарному периоду, имеются ли свободные места и какова стоимость билетов. К числу характерных операций по изменению данных относятся бронирование места на рейс, определение номера места и выбор обеденного меню. В любой момент к одним и тем же элементам данных могут обращаться несколько операторов-кассиров. СУБД обязана обеспечить подобную возможность, но исключить любые потенциальные проблемы, связанные, например, с продажей нескольких билетов на одно место, а также предотвратить опасность потери записей данных, если система внезапно выйдет из строя.

Банковские системы

Базы данных банковских систем содержат информацию об именах и адресах клиентов, лицевых счетах, кредитах, остатках и оборотах денежных средств, а также о связях между элементами бухгалтерской и персональной информации, т.е. о том, кто из клиентов владеет теми или иными счетами, кредитами и т.п. Весьма распространенными являются запросы об остатке на счете, а также операции по изменению его состояния, связанные с приходом или расходом средств.

Как и в ситуации с системой бронирования авиабилетов, вполне естественной выглядит возможность одновременного доступа к информации со стороны многочисленных клиентов и служащих банка, пользующихся локальными терминалами, банкоматами или средствами Web. В этой связи жизненно важное значение приобретает требование о том, чтобы единовременное обращение к одному и тому же банковскому счету ни при каких условиях не приводило к потере отдельных транзакций. Какие бы то ни было ошибки в данном случае совершенно не допустимы. Как только, например, деньги со счета выданы банкоматом, система должна немедленно сохранить информацию о выполненной расходной операции — даже тогда, когда в тот же момент внезапно отключилось электропитание. Соответствующие решения, обладающие требуемым уровнем надежности, далеко не очевидны и могут быть отнесены к разряду “фигур высшего пилотажа” в сфере технологий СУБД.

Корпоративные системы

Многие из ранних приложений баз данных были предназначены для хранения корпоративной информации — записей о продажах и закупках, данных об остатках на счетах внутреннего бухгалтерского учета или персональных сведений о служащих компании (их именах, адресах проживания, уровнях фиксированной заработной платы, надбавках, отчислениях и т.д.). Запросы к таким базам данных позволяют получать информацию о состоянии счетов, выплатах сотрудникам и т.п. Каждая операция купли/продажи, прихода/расхода, приема со-

трудника на работу, его продвижения по службе или увольнения приводит к изменению соответствующих элементов данных.

Самые первые СУБД, во многом унаследовавшие свойства файловых систем, оказывались способными представлять результаты запросов практически только в том виде, который соответствовал структуре хранения данных. В подобных системах баз данных находили применение несколько различных моделей описания структуры информации — в основном, "иерархическая" древовидная и "сетевая" графовая. В конце 1960-х годов последняя получила признание и нашла отражение в стандарте CODASYL (Committee on Data Systems and Languages).[1]

Одной из основных проблем, препятствовавших распространению и использованию таких моделей и систем, было отсутствие поддержки ими высокоуровневых языков запросов. Например, язык запросов CODASYL для перехода от одного элемента данных к другому предусматривал необходимость просмотра вершин графа, представляющего элементы и связи между ними. Совершенно очевидно, что в таких условиях для создания даже самых простых запросов требовались весьма серьезные усилия.

1.1.2. Системы реляционных баз данных

После опубликования в 1970 году знаменитой статьи Э.Ф. Кодда (E.F. Codd)[2] системы баз данных претерпели существенные изменения. Д-р Кодд предложил схему представления данных в виде таблиц, называемых *отношениями* (relations). Структуры таблиц могут быть весьма сложными, что не снижает скорости обработки самых различных запросов. В отличие от ранних систем баз данных, рассмотренных выше, пользователю реляционной базы данных вовсе не требуется знать об особенностях организации хранения информации на носителе. Запросы к такой базе данных выражаются средствами высокоуровневого языка, позволяющего значительно повысить эффективность работы программиста.

Различным аспектам реляционной модели, положенной в основу многих современных СУБД, посвящен материал большинства глав нашей книги, начиная с главы 3 (см. с. 87), в которой изложены основные концепции модели. В главе 6 (см. с. 249) и нескольких следующих главах мы расскажем о языке *SQL* (Structured Query Language — *язык структурированных запросов*) — наиболее важном и мощном представителе семейства языков запросов. Впрочем, дабы продемонстрировать читателю простоту и изящество модели, уже сейчас имеет смысл изложить краткие сведения об отношениях и привести наглядный пример использования SQL, который позволил бы показать, каким образом реляционная модель создает благоприятные условия для применения высокоуровневых запросов и позволяет избежать необходимости "копания" в недрах базы данных.

Пример 1.1. Отношения — это таблицы. Каждый столбец таблицы озаглавлен посредством *атрибута* (attribute), описывающего природу элементов столбца. Например, отношение `Accounts`, содержащее данные о номерах банковских счетов (`accountNo`), их типах (`type`) и величине остатков (`balance`), могло бы выглядеть следующим образом:

accountNo	*balance*	*type*
`12345`	`1000.00`	`savings`
`67890`	`223322.99`	`checking`
`...`	`...`	`...`

[1] *CODASYL Data Base Task Group April 1971 Report*, ACM, New York.

[2] Codd E.F. *A relational model for large shared data banks*, Comm. ACM, **13**:6 (1970), p. 377–387.

Столбцы таблицы озаглавлены следующими атрибутами: `accountNo`, `type` и `balance`. Под строкой атрибутов расположены строки данных, или *кортежи* (tuples). Здесь явно показаны два кортежа отношения, а многоточия в последней строке указывают на тот факт, что отношение может включать большее число кортежей, по одному на каждый счет, открытый в банке. Первый кортеж свидетельствует о том, что на счете с номером `12345` содержится остаток, равный `1000` денежных единиц, и счет относится к категории накопительных (`savings`). Второй кортеж соответствует чековому (`checking`) счету с номером `67890` и остатком `223322.79`.

Пусть необходимо узнать величину остатка на счете с номером 67890. Соответствующий запрос, оформленный по правилам языка SQL, может выглядеть так:

```
SELECT balance
FROM Accounts
WHERE accountNo = 67890;
```

Рассмотрим другой пример. Чтобы получить список номеров накопительных счетов с отрицательным остатком, достаточно выполнить следующий запрос:

```
SELECT accountNo
FROM Accounts
WHERE type = 'savings' AND balance < 0;
```

Разумеется, мы не вправе ожидать, что двух приведенных примеров будет достаточно, чтобы превратить новичка в эксперта по программированию на языке SQL, но они вполне способны донести до читателя смысл инструкции SQL "select–from–where" ("выбрать–из–при_условии"). Каждое из рассмотренных выражений SQL по существу заставляет СУБД выполнить следующие действия:

1) проверить все кортежи отношения `Accounts`, упомянутого в предложении `FROM`;
2) выбрать те кортежи, которые удовлетворяют некоторому критерию, описанному предложением `WHERE`;
3) возвратить в качестве результата определенные атрибуты выбранных кортежей, перечисленные в предложении `SELECT`.

В реальной ситуации система бывает способна "оптимизировать" запрос, чтобы отыскать некоторый более эффективный способ выполнения поставленного задания, — даже в том случае, если размеры отношений, участвующих в запросе, чрезвычайно велики. ❑

До 1990-х годов системы реляционных баз данных занимали главенствующее положение. Однако технологии СУБД продолжают интенсивно развиваться, и на арене с завидной регулярностью появляются новые теоретические и технологические решения проблемы управления данными. Нам хотелось бы быть последовательными, поэтому мы должны уделить внимание некоторым современным тенденциям развития систем баз данных.

1.1.3. Уменьшение и удешевление систем

Изначально СУБД представляли собой крупные и дорогие программные комплексы, ориентированные на использование больших компьютеров. Обширные "размеры" компьютера были необходимым условием работоспособности СУБД, имевших дело с гигабайтами информации. Сегодня многие гигабайты данных способны уместиться на единственном маленьком дисковом устройстве, поэтому вполне закономерным итогом развития технологий стала возможность установки СУБД на персональных компьютерах. Системы баз данных, основанные на реляционной модели, теперь доступны для использования на самых малых машинах и становятся обычным компонентом вычислительных систем — во многом подобно тому, как в свое время та же счастливая участь постигла приложения текстовых процессоров и электронных таблиц.

1.1.4. Тенденции роста систем

Гигабайт — это в действительности далеко не самый большой объем информации. Размеры корпоративных баз данных зачастую исчисляются сотнями гигабайтов. Помимо того, с удешевлением носителей информации человек получает в свое распоряжение новые возможности хранения все возрастающих массивов данных. Например, базы данных, обслуживающие пункты розничной продажи, нередко включают *терабайты* (1 терабайт равен 1000 гигабайт, или 10^{12} байт) информации, фиксирующей каждую операцию продажи на протяжении длительного периода времени, что позволяет более эффективным образом планировать оптовые поставки товаров (подобные вопросы мы осветим в разделе 1.1.7 на с. 38).

Помимо того, современные базы данных способны хранить не только простые элементы данных, такие как целые числа и короткие строки символов. Они позволяют накапливать графические изображения, аудио- и видеозаписи и чрезвычайно крупные фрагменты данных других типов. Например, для хранения видеоматериалов часовой продолжительности воспроизведения требуется участок носителя емкостью в один гигабайт. Базы данных, предназначенные для хранения графической информации, полученной со спутников, могут охватывать *петабайты* (1000 терабайт, или 10^{15} байт) данных.

Задача обслуживания крупных баз данных требует неординарных технологических решений. Базы данных даже относительно скромных размеров сегодня обычно размещают на дисковых массивах, называемых *вторичными устройствами хранения* (secondary storage devices) — в отличие от *оперативной памяти* (main memory), которая служит *"первичным"* хранилищем информации. Нередко говорят и о том, что наиболее существенной особенностью, отличающей СУБД от других программных продуктов, является их способность к долговременному хранению данных на внешних носителях и загрузке требуемых порций информации в оперативную память по мере необходимости. Технологические решения, кратко рассмотренные ниже, позволяют СУБД эффективно справляться с неуклонно возрастающими потоками данных.

Третичные устройства хранения

Для нормального функционирования самых крупных баз данных, используемых в настоящее время, возможностей, предлагаемых обычными дисковыми накопителями, уже не достаточно. Поэтому разрабатываются различные типы *третичных устройств хранения* (tertiary storage devices) данных. Такое устройство, способное к сохранению терабайтов информации, требует гораздо большего времени для доступа к конкретному элементу данных, нежели диск. Если период обращения к данным на диске исчисляется, как правило, 10–20 миллисекундами, на выполнение аналогичной операции третичным устройством хранения может уйти несколько секунд. Третичные устройства хранения предполагают наличие какого-либо роботизированного транспортного средства, способного перемещать носитель, на котором расположен запрошенный элемент данных, к устройству чтения данных.

Функции отдельных носителей данных в третичных устройства хранения могут выполнять компакт-диски (CD) либо диски формата DVD (digital versatile disk). Захват робота передвигается к определенному диску, выбирает его, перемещает к устройству чтения и загружает в последний.

Параллельные вычисления

Способность системы сохранять громадные объемы данных, разумеется, важна, но она вряд ли будет востребована, если производительность такой системы недостаточно высока. Поэтому СУБД, обрабатывающие чрезмерно большие фрагменты информации, нуждаются в реализации решений, позволяющих повысить скорость вычислений. Один из важнейших подходов к проблеме связан с использованием структур *индексов* (indexes) — мы упомянем их в разделе 1.2.2 на с. 40 и рассмотрим более полно в главе 13 (см. с. 583). Другой способ дости-

жения поставленной цели — обработать больше данных в течение промежутка времени заданной продолжительности — предполагает обращение к средствам распараллеливания вычислений. Параллелизм вычислений реализуется в нескольких направлениях.

Поскольку скорость считывания данных, расположенных на определенном диске, заведомо низка и измеряется несколькими мегабайтами в секунду, ее можно увеличить, если использовать множество дисков и выполнять операции чтения данных с них параллельно (такой подход приемлем даже в том случае, если в системе применяется третичное устройство хранения, так как данные, прежде чем поступить в распоряжение СУБД, в любом случае "кэшируются" на дисках). Такие диски могут служить частью специальным образом спроектированной параллельной вычислительной машины либо компонентами распределенной системы, состоящей из нескольких машин, каждая из которых ответственна за поддержку определенного звена базы данных и при необходимости допускает обращение при посредничестве высокопроизводительного сетевого соединения.

Разумеется, способность системы к быстрому перемещению данных — равно как и возможность хранения больших объемов информации — сама по себе еще не служит гарантией того, что результаты запросов будут получены достаточно быстро. Независимо от применяемых аппаратных средств, безусловно необходимы реализации соответствующих алгоритмов, позволяющих "дробить" множества запросов на части таким образом, чтобы позволить параллельным компьютерам или распределенным сетям компьютеров эффективно использовать все имеющиеся вычислительные ресурсы. Параллельное и распределенное управление сверхбольшими базами данных остается сферой активных исследований и технологических разработок; более серьезное внимание этой теме мы уделим в разделе 15.9 на с. 741.

1.1.5. Системы "клиент/сервер" и многоуровневые архитектуры

Огромный пласт современного программного обеспечения поддерживает архитектуру *клиент/сервер* (client/server), в соответствии с которой запросы, сформированные одним процессом (*клиентом*), отсылаются для обработки другому процессу (*серверу*). Системы баз данных в этом смысле — не исключение: обычной практикой является разделение функций СУБД между процессом сервера и одним или несколькими клиентскими процессами.

Простейший вариант архитектуры "клиент/сервер" предполагает, что СУБД целиком представляет собой сервер, за исключением интерфейсов запросов, которые взаимодействуют с пользователем и отсылают запросы и другие команды на сервер с целью их выполнения. Например, в реляционных системах для представления запросов клиентов к серверу используются инструкции языка SQL. Результаты выполнения запросов сервером возвращаются клиентам в форме таблиц, или отношений. Взаимосвязь клиента и сервера может быть гораздо более сложной, особенно в тех случаях, когда результаты выполнения запросов обладают сложной структурой или большим объемом. Более подробно мы рассмотрим этот вопрос в следующем разделе.

Существует и тенденция к тому, чтобы изрядную часть функций оставлять за клиентом, если серверное звено в общей структуре системы является "узким местом" ввиду того, что сервер базы данных из-за большого числа единовременных подключений испытывает чрезмерную нагрузку. В последнее время широкое распространение получили многоуровневые архитектуры, в которых СУБД отводится роль поставщика динамически генерируемого содержимого Web-сайтов. СУБД продолжает действовать как сервер базы данных, но его клиентом теперь служит *сервер приложений* (application server), который управляет подключениями к базе данных, транзакциями, авторизацией и иными процессами. Клиентами серверов приложений в свою очередь могут являться Web-серверы, обслуживающие конечных пользователей, и другие программные приложения.

1.1.6. Данные мультимедиа

Другая важная тенденция развития современных СУБД связана с поддержкой данных мультимедиа. Употребляя термин *мультимедиа* (multimedia), мы имеем в виду информацию, представляющую сигнал определенного вида. К числу наиболее распространенных разновидностей данных мультимедиа относятся аудио- и видеозаписи, сигналы, полученные от радаров, спутниковые изображения, а также документы и графика различных форматов. Общим свойством элементов данных мультимедиа является большой объем, способный изменяться в широких пределах, что отличает их от традиционных единиц представления информации — целых чисел, строк фиксированной длины и т.п.

Потребность в хранении данных мультимедиа способствует развитию технологий СУБД в нескольких направлениях. Например, общепринятые операции, применимые в отношении простых элементов данных, не приемлемы в контексте проблемы обработки информации мультимедиа. Хотя задача поиска в базе данных всех номеров банковских счетов с отрицательным сальдо, предполагающая сравнение значений остатков с вещественным числом 0.0, является вполне правомерной, совершенно не уместной будет выглядеть попытка отыскания в базе данных фотографических изображений лиц, "похожих" на определенную физиономию.

Чтобы обеспечить возможность выполнения манипуляций сложной информацией (например, операций по обработке графических изображений), СУБД должны позволять пользователям определять новые функции, которые необходимы в той или иной ситуации. Расширение функционального потенциала систем зачастую осуществляется на основе объектно-ориентированного подхода — даже в контексте сугубо реляционных СУБД, которые после подобной "доработки" становятся "объектно-реляционными" (object-relational). Аспекты объектно-ориентированного программирования приложений баз данных мы будем рассматривать неоднократно — в частности, в главах 4 (см. с. 149) и 9 (см. с. 419).

Большие размеры объектов данных мультимедиа вынуждают разработчиков СУБД модифицировать функции менеджеров хранения данных, чтобы обеспечить возможность размещения в базе данных объектов или кортежей объемом в гигабайт и более. Одна из серьезных проблем, возникающих вследствие непомерного роста объема элементов данных, связана с доставкой клиенту результатов отправленного им запроса. В контексте традиционной реляционной базы данных результатом выполнения запроса служит набор кортежей, которые возвращаются клиенту как единое целое. А теперь предположим, что результатом выполнения запроса является видеоклип объемом в один гигабайт. Вполне очевидно, что задача доставки клиенту элемента данных размером в гигабайт в виде единого целого не может считаться безусловно корректной. Во-первых, фрагмент данных настолько велик, что попытка его пересылки воспрепятствует обработке сервером всех других запросов, ожидающих обслуживания. Во-вторых, пользователь может быть заинтересован в получении только небольшой части клипа, но он лишен возможности точно сформулировать цель запроса, не взглянув, например, на начальный фрагмент материала. В-третьих, даже в том случае, если пользователь абсолютно убежден в необходимости получения всего клипа, надеясь просмотреть его целиком, вполне достаточно передавать информацию частями с фиксированной скоростью в течение одного часа (промежутка времени, необходимого для воспроизведения гигабайта сжатых видеоданных). Таким образом, подсистема хранения данных СУБД, поддерживающей форматы мультимедиа, должна предлагать интерактивный режим обслуживания запросов, чтобы пользователь смог указать, желает ли он получать фрагменты данных по дополнительному требованию либо непрерывно с фиксированной частотой.

1.1.7. Интеграция информации

По мере повышения роли информации в жизни общества изменяются и развиваются способы использования существующих информационных источников. В качестве примера рас-

смотрим компанию, желающую обладать электронным каталогом всех собственных товаров, чтобы потенциальные клиенты получили возможность просматривать каталог средствами Web-броузера, выбирать нужные товары и оформлять электронные заказы. Крупные компании состоят из многих подразделений. Каждое независимое подразделение вправе создавать собственные базы данных о товарах и применять при этом различные СУБД, различные структуры данных и даже употреблять различные наименования одной и той же вещи либо один термин для обозначения нескольких вещей.

Пример 1.2. Рассмотрим компанию, производящую компьютерные диски и состоящую из нескольких филиалов. В каталоге продукции одного филиала скорость вращения дисков представлена, скажем, в виде значения количества оборотов в *секунду*, а в каталоге другого филиала в качестве единицы измерения того же параметра выбрано количество оборотов в *минуту*. Третий каталог вовсе не содержит упоминаний о скорости вращения дисков. В каталоге филиала, производящего *гибкие* диски, для обозначения продукции употребляется термин "диск". То же название товара используется и в подразделении, занимающемся разработкой *жестких* дисков. В одном случае дорожки диска могут быть названы "треками", а в другом — "цилиндрами". ❑

Строгую централизацию управления не всегда можно считать приемлемым решением рассмотренной проблемы. Подразделения компании, возможно, инвестировали большие средства в создание собственных баз данных еще до того момента, когда на повестке дня возник вопрос о целесообразности интеграции информации в рамках всей компании. Также вполне допустимо, что какое-либо подразделение, обладающее собственной базой данных, вошло в состав компании совсем недавно. Исходя из подобных и иных причин так называемые *унаследованные базы данных* (legacy databases) подразделений не могут (и не должны) непосредственно заменяться единой центральной базой данных компании. Более разумно построить "поверх" существующих унаследованных баз новую информационную структуру, способную представить всю продукцию компании в согласованном и последовательном виде.

Один из самых популярных подходов к решению такой задачи связан с использованием технологий *хранилищ данных* (data warehouses), которые предполагают копирование информации из унаследованных баз данных с соответствующей трансляцией и последующим сохранением в центральной базе данных. При изменении состояния унаследованной базы данных необходимые исправления вносятся и в содержимое хранилища, хотя не обязательно автоматически и немедленно. Весьма часто репликацию данных осуществляют ночью, когда вероятность загрузки унаследованных баз данных наиболее низка.

Унаследованная база данных, таким образом, продолжает выполнять все обычные функции, предусмотренные в период ее проектирования, а новые, такие как поддержка электронных каталогов в Web, возлагаются на хранилище данных. Содержимое хранилищ данных используется также в целях планирования и анализа: аналитики компании получают возможность обращаться к хранилищу данных с запросами, позволяющими, например, выявить тенденции продаж продукции, чтобы оптимизировать ее ассортимент и спланировать дальнейшее развитие производства. Хранилища данных открывают перспективы применения новейших технологий *"разработки"* (или *"добычи"*) *данных* (data mining) — поиска любопытных и необычных образцов информации и использования их для оптимизации бизнес-процессов. Эти и другие вопросы интеграции данных мы обсудим в главе 20 (см. с. 1003).

1.2. Обзор структуры СУБД

На рис. 1.1 приведена структурная схема типичной системы управления базами данных. Прямоугольниками из одинарных линий обозначены компоненты системы, а фигурами из двойных линий — структуры данных, организованные в памяти. Непрерывные отрезки со стрелками указывают направление потоков управляющих инструкций и данных, а пунктир-

ными линиями отмечены только потоки данных. Поскольку схема достаточно сложна, мы будем рассматривать ее поэтапно. Начнем с того, что в верхней части рисунка изображены два различных источника управляющих инструкций, направляемых системе:

1) рядовые пользователи и прикладные программы, запрашивающие или изменяющие данные;
2) *администратор базы данных* (database administrator — DBA) — лицо или группа лиц, ответственных за поддержку и развитие структуры, или *схемы*, базы данных.

1.2.1. Команды языка определения данных

Относительно более простым является множество команд, поступающих от администратора базы данных (их поток изображен в правой верхней части рис. 1.1). В качестве примера рассмотрим базу данных факультета университета. Администратор может включить в ее состав таблицу (отношение), строки (кортежи) которой представляют информацию о студентах — их имена и фамилии, номера курсов и курсовые отметки. Администратор вправе ограничить множество допустимых отметок, выставляемых студентам по окончании курса, скажем, значениями A, B, C, D и F. Структура создаваемой таблицы и вводимая информация об ограничениях становятся частью схемы базы данных. Чтобы выполнить задуманное, администратор должен обладать специальными полномочиями по выполнению команд, затрагивающих схему базы данных, поскольку таковые оказывают самое серьезное влияние на структуру хранения информации. Подобные команды определения данных — их называют командами *языка DDL* (Data Definition Language — *язык определения данных*) — подвергаются лексической обработке с помощью *компилятора DDL* (DDL compiler) и передаются для выполнения *исполняющей машине* (execution engine), которая при посредничестве *менеджера ресурсов* (resource manager) изменяет *метаданные* (metadata — "данные о данных"), т.е. информацию, описывающую схему базы данных.

1.2.2. Обработка запросов

Большая часть обращений к базе данных инициируется пользователями и приложениями (соответствующие потоки команд и данных изображены в левой части рис. 1.1). Подобные действия не оказывают влияния на схему базы данных, но затрагивают содержимое последней (если команда предусматривает внесение изменений) либо предполагают чтение данных (если команда содержит запрос). В разделе 1.1 на с. 32 мы уже писали о том, что такие команды оформляются с помощью *языка управления данными* (Data Manipulation Language — DML), или, проще говоря, *языка запросов* (query language). Существует множество различных языков управления данными, но самым распространенным и мощным из них является SQL, упоминавшийся нами в примере 1.1 (см. с. 34). Инструкции языка DML обрабатываются двумя отдельными подсистемами СУБД, описанными ниже.

Получение ответа на запрос

Запрос анализируется и оптимизируется *компилятором запросов* (query compiler). Сформированный компилятором *план запроса* (query plan), или последовательность действий, подлежащих выполнению системой с целью получения ответа на запрос, передается *исполняющей машине*. Исполняющая машина направляет группу запросов на получение небольших порций данных — как правило, строк (кортежей) таблицы (отношения) — *менеджеру ресурсов*, который "осведомлен" об особенностях размещения информации в *файлах данных* (data files), содержащих таблицы, о форматах и размерах записей в этих файлах и о структурах *индексных файлов* (index files), обеспечивающих существенное ускорение процессов поиска запрошенных данных.

Запросы на получение данных транслируются в адреса страниц и пересылаются *менеджеру буферов* (buffer manager). О роли менеджера буферов мы поговорим ниже, но здесь уместно кратко отметить, что его задачей является обращение к соответствующим порциям данных на носителях вторичных устройств хранения (обычно, дисках), где они располагаются постоянно, с последующим переносом данных в буферы, размещаемые в оперативной памяти, и наоборот. Единицами потоков обмена данными между буферами в памяти и диском являются страница или "дисковый блок".

Чтобы получить информацию с диска, менеджеру буферов приходится обращаться к услугам *менеджера хранения данных* (storage manager), который, решая возложенные на него задачи, может вызывать команды операционной системы, но чаще всего непосредственно инициирует инструкции дискового контроллера.

Обработка транзакций

Запросы и другие команды языка управления данными группируются в *транзакции* (transactions) — процессы, которые должны выполняться *атомарным образом* (atomically) и *изолированно* (in isolation) друг от друга. Зачастую каждый отдельный запрос или операция по изменению данных является самостоятельной транзакцией. Транзакция обязана обладать свойством *устойчивости* (durability). Это значит, что результат каждой завершенной транзакции должен быть зафиксирован в базе данных даже в тех ситуациях, когда после окончания транзакции система по той или иной причине выходит из строя. На рис. 1.1 *процессор транзакций* (transaction processor) представлен в виде двух основных компонентов:

1) *планировщика заданий* (scheduler), или *менеджера параллельных заданий* (concurrency-control manager), ответственного за обеспечение атомарности и изолированности транзакций;
2) *менеджера протоколирования и восстановления* (logging and recovery manager), гарантирующего выполнение требования устойчивости транзакций.

Указанные компоненты будут подробно рассмотрены в разделе 1.2.4 на с. 42.

1.2.3. Менеджеры буферов и хранения данных

Информация, хранимая в базе данных, обычно располагается на носителях вторичных устройств хранения. Термин *вторичное устройство хранения* (secondary storage device), употребляемый применительно к современной компьютерной системе, обычно означает магнитный диск. Однако, чтобы выполнить любую сколько-нибудь полезную операцию над данными, необходимо перенести их в оперативную память. Задача управления размещением информации на диске и обмена ею между диском и оперативной памятью возлагается на *менеджер хранения данных* (storage manager).

В простой системе баз данных роль менеджера хранения может быть поручена непосредственно файловой системе, обслуживаемой соответствующей операционной системой. Впрочем, из соображений обеспечения эффективности вычислений СУБД обычно (по меньшей мере, при определенных обстоятельствах) самостоятельно управляет размещением данных на дисках. В ответ на запрос со стороны менеджера буферов менеджер хранения данных определяет положение файлов на диске и получает набор блоков диска, содержащих требуемый файл или его часть. Отметим, что поверхность диска обычно разделена на *дисковые блоки* (disk blocks) — непрерывные участки, способные сохранять большое число байтов информации, до 2^{12} или 2^{14} (4096 или 16384).

Менеджер буферов (buffer manager) ответствен за разбиение доступной оперативной памяти на буферы (buffers) — участки-*страницы* (pages), куда может быть помещено содержимое дисковых блоков. Все компоненты СУБД, обращающиеся к дисковой информации, взаимодействуют с буферами и менеджером буферов — либо непосредственно, либо при помощи ис-

полняющей машины. Данные, требуемые компонентами системы, могут относиться к одной из следующих категорий:

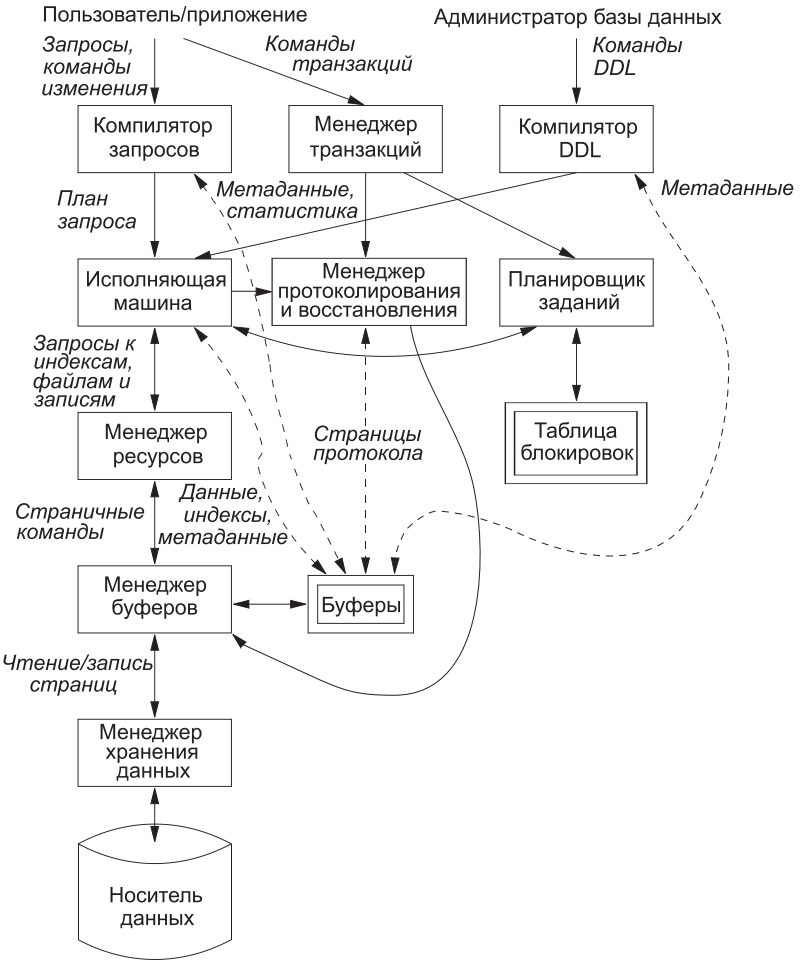

Рис. 1.1. Компоненты системы управления базами данных

1) *собственно данные* — содержимое базы данных как таковой;
2) *метаданные* — описание схемы, или логической структуры, базы данных, введенных ограничений и т.д.;
3) *статистика* — информация о свойствах данных, таких как размеры различных отношений, о вероятностных распределениях хранящихся значений и т.п., собранная СУБД;
4) *индексы* — структуры данных, обеспечивающие эффективный доступ к информации в базе данных.

За подробными сведениями о функциональных особенностях менеджера буферов и его значении в общей структуре СУБД обращайтесь к разделу 15.7 на с. 732.

1.2.4. Обработка транзакций

Обычной практикой является оформление одной или нескольких операций над базой данных в виде *транзакции* (transaction) — единицы работы, которая должна быть выполнена *атомарным образом* и *изолированно* от других транзакций. Кроме того, СУБД обязана удовлетворять требованию *устойчивости* транзакций: результат выполнения завершенной транзакции не должен быть утрачен ни при каких условиях. *Менеджер транзакций* (transaction manager) воспринимает от приложения *команды транзакций* (transaction commands), которые свидетельствуют о начале и завершении транзакции, а также передают информацию о предпочтениях приложения в отношении параметров транзакции (приложение, например, может отказаться от свойства атомарности транзакции). *Процессор транзакций* (transaction processor) выполняет функции, рассмотренные ниже.

1. *Протоколирование* (logging). С целью удовлетворения требования *устойчивости* (durability) транзакций каждое изменение в базе данных фиксируется в специальных дисковых файлах. *Менеджер протоколирования* (logging manager) в своей работе руководствуется одной из нескольких стратегий, призванных исключить вредные последствия системных сбоев во время выполнения транзакции, а *менеджер восстановления* (recovery manager) в случае возникновения подобных ситуаций способен считать протокол изменений и привести базу данных в некоторое сообразное состояние. Информация протокола изначально сохраняется в буферах; затем менеджер протоколирования в определенные моменты времени взаимодействует с менеджером буферов, дабы убедиться, что содержимое буферов действительно сохранено на диске (где данные находятся под надежной защитой в момент возможного краха системы).
2. *Управление параллельными заданиями* (concurrency control). Транзакции обязаны выполняться в полной изоляции друг от друга. Горькая истина, однако, заключается в том, что в реальных системах одновременно могут действовать несколько процессов-транзакций. *Планировщик заданий* (scheduler), или *менеджер параллельных заданий* (concurrency-control manager), должен обеспечить такой режим работы системы, чтобы результат выполнения отдельных перемежающихся во времени операций многочисленных транзакций оказался таким, как если бы транзакции в действительности инициировались, протекали и полностью завершались в строгой очередности, не “пересекаясь” одна с другой. Типичный планировщик заданий добивается поставленной перед ним цели, устанавливая признаки *блокировки* (lock) на соответствующие фрагменты содержимого базы данных. Блокировки препятствуют возможности единовременного обращения нескольких транзакций к порции данных такими способами, которые плохо согласуются друг с другом. Признаки блокировки обычно хранятся в *таблице блокировок* (lock table), размещаемой в оперативной памяти, как показано на рис. 1.1 (см. с. 42). Планировщик заданий воздействует на процесс выполнения запросов и других операций, запрещая исполняющей машине обращаться к блокированным порциям данных.
3. *Разрешение взаимоблокировок* (deadlock resolution). Поскольку транзакции состязаются за ресурсы, которые могут быть блокированы планировщиком заданий, возможно возникновение таких обстоятельств, когда ни одна из транзакций не в состоянии продолжить работу ввиду того, что ей необходим ресурс, находящийся в ведении другой транзакции. Менеджер транзакций обладает прерогативой вмешиваться в ситуацию и прерывать (*откатывать* — “rollback”) одну или несколько транзакций, чтобы позволить остальным продолжить работу.

1.2.5. Процессор запросов

Подсистема, в наибольшей степени определяющая показатели производительности СУБД, видимые, что называется невооруженным глазом, носит название *процессора запросов* (query processor). На рис. 1.1 (см. с. 42) процессор запросов представлен двумя компонентами, рассмотренными ниже.

ACID: свойства транзакций

Принято говорить, что правильным образом реализованные транзакции удовлетворяют "условиям *ACID*":

- *A* представляет свойство "atomicity" (*атомарность*) — транзакция осуществляется в соответствии с принципом "все или ничего", т.е. должна быть выполнена либо вся целиком, либо не выполнена вовсе;
- *I* служит сокращением термина "isolation" (*изолированность*) — процесс протекает таким образом, словно в тот же период времени других транзакций не существует;
- *D* определяет требование под названием "durability" (*устойчивость*) — результат выполнения завершенной транзакции не должен быть утрачен ни при каких условиях.

Оставшаяся буква аббревиатуры, *C*, обозначает свойство "consistency" (*согласованность*). Все базы данных вводят те или иные *ограничения* (constraints), обусловливающие особенности различных элементов данных и их взаимную согласованность (так, например, сумма бухгалтерской проводки не может быть отрицательной). Транзакции обязаны сохранять согласованность данных. О том, каким образом ограничения определяются в схеме базы данных, мы расскажем в главе 7 (см. с. 317), а вопросы, касающиеся поддержки системой требования согласованности данных, будут освещены в разделе 18.1 на с. 880.

1. *Компилятор запросов* (query compiler) транслирует запрос во внутренний формат системы — *план запроса* (query plan). Последний описывает последовательность инструкций, подлежащих выполнению. Часто инструкции плана запроса представляют собой реализации операций "реляционной алгебры" (мы расскажем о ней в разделе 5.2 на с. 205). Компилятор запросов состоит из трех основных частей:
   a) *синтаксического анализатора запросов* (query parser), создающего на основе текста запроса древовидную структуру данных;
   b) *препроцессора запросов* (query preprocessor), выполняющего семантический анализ запроса (проверку того, все ли отношения и их атрибуты, упомянутые в тексте запроса, действительно существуют) и функции преобразования дерева, построенного анализатором, в дерево алгебраических операторов, отвечающих исходному плану запроса;
   c) *оптимизатора запросов* (query optimizer), осуществляющего трансформацию плана запроса в наиболее эффективную последовательность фактических операций над данными.

   Компилятор запросов, принимая решение о том, какая из последовательностей операций с большей вероятностью окажется самой оптимальной по быстродействию, пользуется метаданными и статистической информацией, накопленной СУБД. Например, наличие *индекса* (index) — специальной структуры данных, обслуживающей процессы доступа к информации отношений посредством хранения определенных значений, которые соответствуют порциям содержимого отношения, — способно существенным образом повлиять на выбор наиболее эффективного плана.

2. *Исполняющая машина* (execution engine) несет ответственность за осуществление каждой из операций, предусмотренных выбранным планом запроса. В процессе своей работы она взаимодействует с большинством других компонентов СУБД — либо напрямую, либо при посредничестве буферов данных. Чтобы получить возможность обрабатывать данные, исполняющая машина обязана считать их с носителя и перенести в буферы. При этом ей необходимо "общаться" с планировщиком заданий, чтобы избежать опасности обращения к блокированным порциям информации, а также с менеджером протоколирования, обеспечивающим гарантии того, что все изменения, внесенные в базу данных, должным образом зафиксированы в протоколе.

1.3. Обзор технологий СУБД

Вопросы, имеющие отношение к проблематике создания систем баз данных, можно поделить на три категории, описанные ниже.

1. *Проектирование баз данных.* Как разрабатывать полезные базы данных? Информация каких разновидностей должна быть включена в базу данных? Как надлежит структурировать данные? Какие предположения следует выдвигать относительно типов и значений элементов данных? Как обеспечить взаимосвязь этих элементов?
2. *Программирование приложений баз данных.* Каковы средства представления запросов и других операций над содержимым базы данных? Каким образом следует использовать другие возможности СУБД, такие как транзакции или ограничения, в конкретном программном приложении? Как соотносится и сочетается программирование баз данных с программированием обычных приложений?
3. *Реализация систем баз данных.* Что необходимо сделать, чтобы реализовать конкретную СУБД, включая и такие ее функции, как обработка запросов, управление транзакциями и эффективная организация хранения данных?

1.3.1. Проектирование баз данных

В главе 2 на с. 51 мы рассмотрим абстракцию процесса проектирования базы данных, называемую *ER-моделью*, или *моделью "сущность–связь"* (entity-relationship model). Глава 3 на с. 87 познакомит вас с *реляционной моделью* (relational model), лежащей в основе большинства наиболее распространенных коммерческих СУБД (некоторые особенности реляционной модели мы кратко освещали в разделе 1.1.2 на с. 34). Мы покажем, как конкретный образец ER-модели транслируется в соответствующую реляционную схему базы данных. Позже, в разделе 6.6 на с. 296, будут изложены сведения о том, как реляционная схема базы данных может быть описана средствами подмножества языка SQL, которое имеет отношение к определению данных.

В главе 3 рассматривается и понятие *зависимостей* (dependencies), формально выражающее предположения о взаимоотношениях кортежей отношения. Использование аппарата определения зависимостей позволяет улучшить качество проекта реляционной базы данных в ходе процесса, известного как *нормализация* (normalization) отношений.

В главе 4 на с. 149 мы изложим сведения об объектно-ориентированных подходах к проектированию баз данных. Здесь рассмотрены особенности языка *ODL* (Object Definition Language — *язык определения объектов*), позволяющего описывать базы данных высокоуровневыми объектно-ориентированными средствами. Мы расскажем также о способах сочетания инструментов объектно-ориентированного проектирования с реляционным моделированием, что подведет вас к пониманию "объектно-реляционной" (object-relational) парадигмы. В главе 4 затрагиваются и вопросы использования *"полуструктурированных" данных*

(semistructured data) — особенно гибкой модели представления информации — и современной реализации этой модели в виде языка описания документов XML.

1.3.2. Программирование приложений баз данных

В главах 5–10 на с. 203–487 излагаются основы программирования приложений, ориентированных на использование баз данных. Главу 5 мы начнем с рассмотрения абстрактной трактовки запросов в рамках реляционной модели, введя семейство операторов, применяемых в контексте отношения, которые составляют существо *реляционной алгебры* (relational algebra).

Как создаются индексы

Вы, читатель, возможно, уже изучали курс структур данных и осведомлены о том, что *хеш-таблицы* (hash tables) — это весьма многообещающее средство создания индексов. В ранних СУБД хеш-таблицы использовались особенно интенсивно. Но сегодня наиболее распространенной структурой данных, применяемой для создания индексов, является *B-дерево* (B-tree; буква *B* служит сокращением термина "balanced" — *сбалансированное*). Структура B-дерева представляет собой обобщение так называемого *сбалансированного бинарного дерева поиска* (balanced binary search tree): если вершина бинарного дерева обладает, самое большее, двумя дочерними вершинами, вершина B-дерева может иметь произвольное число дочерних вершин. Если принять во внимание, что данные B-дерева обычно размещаются на диске, а не в оперативной памяти, структура проектируется таким образом, чтобы информация о каждой ее вершине занимала отдельный блок диска. Поскольку обычный размер блока достигает 2^{12} (4096) байт, в одном блоке вполне возможно хранить данные о сотнях дочерних вершин дерева. Таким образом, при поиске данных по B-дереву необходимость обращения к более чем нескольким блокам возникает сравнительно редко.

Трудоемкость дисковых операций пропорциональна количеству запрашиваемых блоков диска. Поэтому поиск по B-дереву, затрагивающий, как правило, всего несколько блоков диска, существенно более эффективен, нежели бинарный поиск, в процессе выполнения которого приходится посещать вершины дерева, относящиеся к многим различным блокам. Указанное различие между деревьями бинарного поиска и B-деревьями — это один из характерных примеров того, что структуры данных, наиболее подходящие для хранения данных в оперативной памяти, не обязательно столь же эффективны, если данные располагаются на диске.

Главы 6–8 на с. 249–418 посвящены технологиям программирования на языке *SQL*. Как мы говорили прежде, SQL преобладает на современном рынке языков запросов. Глава 6 познакомит вас с базовыми понятиями, касающимися представления запросов и схем баз данных на языке SQL. Глава 7 на с. 317 охватывает аспекты SQL, связанные с заданием *ограничений* (constraints) и построением *триггеров* (triggers).

Глава 8 на с. 349 дает ответы на некоторые более сложные вопросы применения SQL. Простейшая модель программирования на SQL предусматривает обособленное употребление элементов "чистого" интерфейса языка, но в большинстве реальных ситуаций фрагменты SQL-кода используются в контексте крупных программных проектов, написанных на традиционных языках, таких как C. В главе 8 мы расскажем о том, как правильно применять выражения SQL внутри кода других программ, а также переносить информацию из базы данных в переменные программы и наоборот. В той же главе приводятся сведения об использовании в прикладных программах механизмов поддержки транзакций, подключения клиентов к сер-

верам и авторизации доступа к информации со стороны пользователей, не являющихся владельцами объектов базы данных.

В главе 9 на с. 419 мы сосредоточим внимание на двух направлениях объектно-ориентированного программирования приложений баз данных. Первое из них связано с применением языка *OQL* (Object Query Language — *язык объектных запросов*), появление которого можно рассматривать как тенденцию к пополнению C++ и других объектно-ориентированных языков общего назначения инструментами, удовлетворяющими требованиям концепции высокоуровневого программирования с ориентацией на использование баз данных. Второе направление, имеющее отношение к недавно принятым нововведениям, которые касаются объектно-ориентированных "расширений" стандарта SQL, выглядит, с другой стороны, как попытка обеспечения совместимости реляционной парадигмы и языка SQL с моделью объектно-ориентированного программирования.

Наконец, в главе 10 на с. 453 мы вновь вернемся к теме абстрактных языков запросов, рассмотрение которой начато в главе 5 (см. с. 203). Мы расскажем о *логических языках запросов* (logical query languages) и об использовании их с целью расширения возможностей современных диалектов SQL.

1.3.3. Реализация систем баз данных

Третья часть книги посвящена технологиям реализации СУБД, которые можно условно поделить на три категории, перечисленные ниже.

1. *Управление процессами хранения данных* — использование вторичных устройств хранения для обеспечения эффективного доступа к информации.
2. *Обработка запросов* — оптимальное обслуживание запросов, оформленных средствами высокоуровневых языков, подобных SQL.
3. *Управление транзакциями* — поддержка транзакций, обеспечивающая выполнение условий ACID (см. раздел 1.2.4 на с. 42).

Каждой из названных тем отведено несколько глав книги.

Управление процессами хранения данных

В главе 11 на с. 489 рассматривается иерархия устройств памяти. Поскольку вторичные устройства хранения данных, в частности диски, являются наиболее важным средством сбережения информации в рамках СУБД, мы уделим повышенное внимание способам хранения данных на диске и доступа к ним. Вашему вниманию будет предложена "блоковая модель" размещения данных на диске, оказывающая влияние почти на все стороны "деятельности" СУБД.

В главе 12 на с. 549 обсуждаются вопросы хранения отдельных элементов данных — отношений, кортежей, значений атрибутов и равнозначных им сущностей, принятых в других моделях представления информации, — в соответствии с требованиями блоковой модели дисковых данных. Далее мы уделим внимание основным структурам данных, находящим применение при конструировании индексов. Уместно напомнить, что индекс — это структура данных, обеспечивающая эффективный доступ к информации на диске. Глава 13 на с. 583 содержит сведения о важных *одномерных* (one-dimensional) индексных структурах данных — *последовательных файлах* (sequential files), *B-деревьях* (B-trees) и *хеш-таблицах* (hash tables). Подобные индексы широко используются в СУБД для оптимизации запросов, предполагающих поиск группы кортежей, которые удовлетворяют заданному значению некоторого атрибута. B-деревья применяются также для доступа к содержимому отношения, отсортированного в порядке убывания или возрастания определенного атрибута. В главе 14 на с. 637 рассматриваются *многомерные* (multidimensional) индексы,

представляющие собой структуры данных, используемые при создании специализированных разновидностей баз данных (например, предназначенных для хранения географической информации), запросы к которым обычно предполагают поиск элементов данных, отвечающих нескольким критериям. Подобные индексные структуры хорошо приспособлены для удовлетворения сложных SQL-запросов, в которых ограничения накладываются на целую группу атрибутов, и некоторые из таких структур уже обрели поддержку со стороны ряда коммерческих СУБД.

Обработка запросов

В главе 15 на с. 683 излагаются сведения о процессах выполнения запросов. Мы представим вашему вниманию большое количество алгоритмов, позволяющих эффективно реализовать множество операций реляционной алгебры. Алгоритмы спроектированы таким образом, чтобы оптимизировать действия с дисковыми данными, и в некоторых случаях довольно существенно отличаются от аналогов, предназначенных для обработки информации, размещенной в оперативной памяти.

Глава 16 на с. 753 содержит сведения об архитектуре *компилятора запросов* (query compiler) и *оптимизатора запросов* (query optimizer). Сначала мы рассмотрим процессы лексического анализа запросов и их семантической проверки. Затем будет рассказано о преобразовании запросов SQL в наборы инструкций реляционной алгебры и критериях выбора *логического плана запроса* (logical query plan), т.е. алгебраического выражения, которое представляет определенные операции, подлежащие выполнению, и необходимые ограничения, затрагивающие порядок следования операций. Наконец, мы обсудим вопросы конструирования *физического плана запроса* (physical query plan), который, учитывая конкретный порядок выполнения операций, задает алгоритм реализации каждой операции.

Управление транзакциями

В главе 17 на с. 837 мы ответим на вопросы, каким образом СУБД обеспечивает *устойчивость* (durability) транзакций. Основной подход к решению проблемы связан с ведением протокола, регистрирующего все изменения в базе данных. При сбое системы (возникающем, например, из-за отключения электропитания) информация, расположенная в оперативной памяти, но не зафиксированная на диске, будет утрачена. Поэтому весьма важно, чтобы операции по перемещению данных из буферов памяти на диск выполнялись своевременно и в соответствующем порядке и чтобы изменения, затрагивающие непосредственно базу данных и протокол, вносились синхронно. Существует несколько стратегий ведения протоколов, но каждый из них каким-либо образом ограничивает свободу действий над данными.

В главе 18 на с. 879 мы затронем тему управления параллельным доступом к данным, способного обеспечить поддержку свойств *атомарности* (atomicity) и *изолированности* (isolation) транзакций. Транзакции представляются в виде последовательностей операций чтения и записи элементов содержимого базы данных. Основное внимание в главе уделено тому, как следует обращаться с *блокировками* (locks) элементов данных. Здесь рассмотрены различные типы блокировок и допустимые методы установки блокировок параллельными транзакциями и освобождения их. Помимо того, мы предоставим сведения о способах обеспечения свойств атомарности и изолированности транзакций без использования механизмов блокирования.

Глава 19 на с. 949 завершает обсуждение проблем обработки транзакций. Мы расскажем о взаимном соотношении требований протоколирования, рассмотренных в главе 17 на с. 837, и условий, обеспечивающих параллельное выполнение транзакций (этой теме посвящена глава 18 на с. 879). Особое внимание мы уделим технологии *разрешения взаимоблокировок* (deadlock resolution) — одной из важнейших функций менеджера транзакций. В главе 19 также изложены сведения об управлении параллельными транзакциями в условиях распределенной вы-

числительной среды (distributed environment) и инструментах обслуживания *длинных* (“long”) транзакций, продолжительность выполнения которых измеряется часами или даже сутками вместо привычных секунд или долей секунды. Длинная транзакция не в состоянии блокировать элементы данных, не причиняя ущерба интересам других потенциальных пользователей тех же данных, что приводит к необходимости переосмысления подходов к проектированию приложений, предусматривающих применение транзакций.

1.3.4. Интеграция информации

Немалая часть современных исследований в сфере технологий баз данных направлена на обеспечение возможностей сочетания в единое целое различных *источников данных* (data sources), среди которых могут быть как привычные базы данных, так и информационные ресурсы, не относящиеся к ведению СУБД. Подобные проблемы кратко освещались в разделе 1.1.7 на с. 38. Заключительная, 20-я, глава (см. с. 1003) содержит сведения о важнейших аспектах интеграции данных. Здесь мы расскажем об основных режимах интеграции, включая использование транслированных и объединенных копий источников информации в рамках *хранилищ данных* (data warehouses) и создание *виртуальных баз данных* (virtual databases) на основе групп источников информации, которое выполняется средствами программных компонентов, называемых *медиаторами* (mediators).

1.4. Резюме

- *Системы управления базами данных.* СУБД отличаются способностью к эффективному выполнению операций обработки больших массивов информации, хранимых на протяжении длительных периодов времени. В СУБД реализована поддержка мощных языков запросов и устойчивых транзакций, которые могут выполняться атомарным образом и независимо от других параллельных транзакций.
- *СУБД и файловые системы.* Традиционные файловые системы не являются полноценной альтернативой СУБД, поскольку они не в состоянии обеспечить реализацию высокопроизводительных функций поиска, возможность эффективной модификации небольших элементов информации, поддержку сложных запросов, гибкую буферизацию требуемых данных в оперативной памяти и атомарное и изолированное выполнение транзакций.
- *Системы реляционных баз данных.* Большинство современных СУБД основано на реляционной модели представления данных, в соответствии с которой информация организуется в виде таблиц. В подобных системах в качестве языка запросов наиболее часто применяется SQL.
- *Вторичные и третичные устройства хранения данных.* Крупные базы данных размещают на вторичных устройствах хранения (как правило, дисках). Наиболее обширные базы данных требуют использования третичных устройств хранения, которые на несколько порядков более емки, но во много раз менее производительны.
- *Системы “клиент/сервер”.* СУБД обычно поддерживают архитектуру “клиент/сервер”, предусматривающую размещение основных компонентов системы на сервере и предоставляющую пользователю необходимый клиентский интерфейс.
- *Языки баз данных.* Существует ряд языков или языковых компонентов для определения структур данных (языки определения данных) и описания запросов и инструкций по изменению элементов информации (языки управления данными).

- *Компоненты СУБД.* К числу наиболее важных компонентов системы управления базами данных относятся менеджер хранения данных, процессор запросов и менеджер транзакций.
- *Менеджер хранения данных* — компонент СУБД, ответственный за размещение и хранение на диске основной информации базы данных, метаданных (сведе-ний о схеме, или логической структуре, данных), индексов (структур, ускоряющих доступ к информации) и протоколов (записей, фиксирующих изменения в базе данных). Важнейшей частью подсистемы хранения данных является менеджер буферов, обеспечивающий перенос порций дисковой информации в буферы оперативной памяти и обратно.
- *Процессор запросов* — элемент СУБД, осуществляющий лексический и семантический разбор запроса, его оптимизацию, выбор соответствующего плана запроса и последующее выполнение плана применительно к реальным данным.
- *Менеджер транзакций* — часть СУБД, реализующая функции ведения протокола изменений с целью обеспечения возможности восстановления данных после сбоев системы, а также управляющая процессами протекания параллельных транзакций и гарантирующая их атомарность (транзакция выполняется либо целиком и до конца, либо не выполняется вовсе) и изолированность (каждая транзакция обслуживается таким образом, словно других конкурирующих транзакций не существует).
- *Будущее систем баз данных.* Основные тенденции в развитии СУБД связаны с поддержкой сверхбольших объектов мультимедиа-данных (таких как видео- или аудиозаписи) и интеграцией информации из многих разнородных источников в единую базу данных.

1.5. Литература

Сегодня, в эпоху существования электронных библиографических каталогов, открытых для доступа с помощью средств Internet и содержащих, в частности, ссылки на все свежие информационные источники, относящиеся к сфере технологий баз данных, совершенно нецелесообразно пытаться привести на страницах книги исчерпывающий список литературы, достойной цитирования. Здесь уместно упомянуть только те печатные работы, которые важны с точки зрения исторической перспективы, указать основные гиперссылки на соответствующие ресурсы Web и отметить некоторые полезные обзоры. Один из каталогов, охватывающих библиографию результатов исследований по проблемам управления базами данных, в свое время был создан и содержится в актуальном состоянии Михелем Леем (Michael Ley) [5]. Альф-Кристиан Ахилл (Alf-Christian Achilles) поддерживает каталог каталогов информации, относящихся к предметной области систем баз данных [1].

Хотя существенные успехи были достигнуты при реализации многих проектов СУБД, наибольшую известность получили два из них — System R, осуществленный в исследовательском центре IBM Almaden Research Center [3], и INGRES, выполненный в Калифорнийском университете в Беркли [7]. Каждый из проектов предусматривал создание систем реляционных баз данных и способствовал становлению этой разновидности систем в качестве главного участника рынка технологий баз данных. Большое количество результатов исследований нашло отражение в обзоре [6].

Одним из самых свежих в серии отчетов по результатам научных изысканий и технологических разработок в области СУБД является отчет [4], который содержит ссылки на многие более ранние информационные источники подобного рода.

За дополнительными сведениями по теории систем баз данных, не нашедшими отражения в литературе, названной выше, обращайтесь к работам [2], [8] и [9].

1. http://liinwww.ira.uka.de/bibliography/Database.
2. Abiteboul S., Hull R., and Vianu V. *Foundations of Databases*, Addison-Wesley, Reading, MA, 1995.
3. Astrahan M.M. et al. *System R: a relational approach to database management*, ACM Trans. on Database Systems, **1**:2 (1976), p. 97–137.
4. Bernstein P.A. et al. *The Asilomar report on database research* (http://www.acm.org/sigmod/record/issues/9812/asilomar.html).
5. http://www.informatik.uni-trier.de/~ley/db/index.html. Адрес зеркального сайта: http://www.acm.org/sigmod/dblp/db/index.html.
6. Stonebraker M., Hellerstein J.M. (eds.) *Readings in Database Systems*, Morgan-Kaufmann, San Francisco, 1998.
7. Stonebraker M., Wong E., Kreps P., and Held G. *The design and implementation of INGRES*, ACM Trans. on Database Systems, **1**:3 (1976), p. 189–222.
8. Ullman J.D. *Principles of Database and Knowledge-Base Systems, Volume I*, Computer Science Press, New York, 1988.
9. Ullman J.D. *Principles of Database and Knowledge-Base Systems, Volume II*, Computer Science Press, New York, 1989.

Глава 2

Модель данных “сущность-связь”

Процесс проектирования базы данных начинается с анализа того, какого рода информация должна быть в ней представлена и каковы взаимосвязи между элементами этой информации. Структура, или *схема* (schema), базы данных определяется средствами различных языков или систем обозначений, пригодных для описания проектов. По завершении этапа уточнений и согласований проект преобразуется в форму, которая может быть воспринята СУБД, и база данных начинает собственную жизнь.

В книге рассматривается несколько систем описания проектов баз данных. Эта глава посвящена изучению *ER-модели*, или *модели “сущность–связь”* (entity-relationship model), ставшей традиционной и наиболее популярной. По своей природе модель является графической: прямоугольники отображают элементы данных, а линии (возможно, со стрелками) указывают на связи между ними.

В главе 3 на с. 87 мы сосредоточим внимание на *реляционной модели* (relational model), представляющей данные об окружающем мире в виде набора таблиц. Множество структур, которые могут быть описаны средствами реляционной модели, разумеется, ограниченно, но этот факт не способен преуменьшить ее значение: модель чрезвычайно проста и плодотворна и является основой большинства современных коммерческих СУБД. Процесс проектирования базы данных обычно начинается с разработки ее схемы посредством инструментов, предлагаемых ER-моделью либо некоторой объектной моделью, с последующей трансляцией схемы в реляционную модель, подлежащую физической реализации.

Альтернативные модели описания данных рассмотрены в главе 4 на с. 149. Раздел 4.2 на с. 152 предлагает введение в язык *ODL* (Object Definition Language — *язык определения объектов*) — стандарт средств описания объектно-ориентированных баз данных. Затем мы проследим, как идеи объектно-ориентированного проектирования, сочетаясь с реляционной моделью представления данных, трансформируются в модель, называемую *объектно-реляционной* (object-relational model).

В разделе 4.6 на с. 187 описан другой подход к моделированию, основанный на концепции *“полуструктурированных” данных* (semistructured data). Последняя предоставляет неограниченную свободу выбора структурных форм, в которые может быть облечена информация. В разделе 4.7 на с. 192 мы обсудим стандарт XML, позволяющий моделировать данные в виде иерархически структурированных документов, используя “тэги” (подобные тэгам HTML),

которые определяют роль и функции текстовых элементов. XML представляет собой блестящее практическое воплощение модели "полуструктурированных" данных.

Рис. 2.1 иллюстрирует, каким образом ER-модели используются при проектировании баз данных. Обычно принято начинать с изучения *понятий и описаний* информации, подлежащей моделированию, а затем пытаться отобразить их в рамках ER-модели. Затем *ER-проект* преобразуется в *реляционную схему*, выраженную средствами языка определения данных для конкретной СУБД. В большинстве случаев СУБД основываются на реляционной модели. Если дело обстоит именно так, в ходе довольно прямолинейного процесса, детали которого мы обсудим в разделе 3.2 на с. 91, абстракция обретает конкретную осязаемую форму, называемую *реляционной схемой базы данных* (relational database schema).

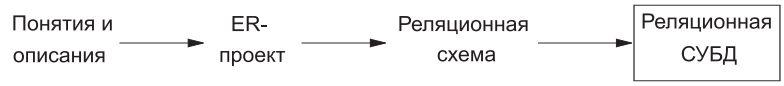

Рис. 2.1. Процесс моделирования и реализации базы данных

Важно отметить, что в то время как в СУБД подчас находят применение модели, отличные от реляционных или объектно-реляционных, систем баз данных, способных реализовать ER-модель непосредственно, попросту не существует. Причина заключается в том, что эта модель недостаточно удовлетворительно согласовывается с эффективными структурами данных, на основе которых должна создаваться "реальная" база данных.

2.1. Элементы ER-модели

Наиболее распространенным средством абстрактного представления структур баз данных является *ER-модель*, или *модель "сущность–связь"* (entity-relationship model). В ER-модели структура данных отображается графически, в виде *диаграммы сущностей и связей* (entity-relationship diagram), состоящей из элементов трех основных типов:

a) *множеств сущностей*;

b) *атрибутов*;

c) *связей*.

Ниже подробно рассмотрен каждый из типов элементов диаграммы сущностей и связей.

2.1.1. Множества сущностей

Сущность (entity) — это абстрактный объект определенного вида. Набор однородных сущностей образует *множество сущностей* (entity set). Понятие сущности обладает определенным сходством с понятием *объекта* (object) (если трактовать последнее так, как это принято делать в объектно-ориентированном проектировании). Примерно таким же образом соотносятся множество сущностей и *класс объектов*. ER-модель, однако, отображает *статические* объекты — она имеет дело со *структурами* данных, но не с *операциями* над данными. Поэтому предполагать, что в ней могут содержаться описания неких "методов", соответствующих множествам сущностей и аналогичных методам класса, нет никаких оснований.

Пример 2.1. На протяжении многих глав книги мы будем рассматривать и развивать пример, касающийся базы данных о кинофильмах, участвующих в них актерах, студиях, осуществивших съемку, и т.п. Каждый из фильмов представляет собой сущность, а коллекция всех фильмов образует множество сущностей. Актеры, снимающиеся в фильмах, также являются сущностями, но другого вида, и их множество — это множество сущностей. Киностудия —

это сущность еще одного вида, а перечень киностудий формирует третье множество сущностей, которое будет использоваться в дальнейших примерах. ❑

Разновидности ER-модели

В некоторых версиях ER-модели атрибуты могут относиться к следующим типам:

1) атомарный, как в версии, рассматриваемой нами;
2) "struct", как в языке C, или кортеж с фиксированным числом атомарных компонентов;
3) множество значений одного типа — атомарного либо "struct".

Например, в качестве типа атрибута в подобной модели может быть задано множество пар, каждая из которых состоит из целого числа и строки.

2.1.2. Атрибуты

Множеству сущностей отвечает набор *атрибутов* (attributes), являющихся свойствами сущностей множества. Например, множеству сущностей *Movies* ("кинофильмы") могут быть поставлены в соответствие такие атрибуты, как *title* ("название") и *length* ("продолжительность" — значение периода времени воспроизведения, выраженное в минутах). В версии ER-модели, рассматриваемой в этой книге, мы предполагаем, что атрибуты представляют собой атомарные значения (например, строки, целые или вещественные числа и т.д.). Существуют и такие варианты модели, в которых понятие типа атрибута трактуется иным образом (см. врезку "Разновидности ER-модели", приведенную выше).

2.1.3. Связи

Связи (relationships) — это соединения между двумя или бо́льшим числом множеств сущностей. Если, например, *Movies* ("кинофильмы") и *Stars* ("актеры") — это два множества сущностей, вполне закономерно наличие связи *Stars-in* ("актеры-участники", снявшиеся в фильме), которая соединяет множества сущностей *Movies* и *Stars*: сущность *m* множества *Movies* соединена с сущностью *s* множества *Stars* посредством связи *Stars-in*, если актер *s* снялся в фильме *m*. Хотя наиболее распространена разновидность *бинарных связей* (binary relationships), соединяющих *два* множества сущностей, ER-модель допускает наличие связей, охватывающих произвольное количество множеств сущностей. Обсуждение вопросов, касающихся *многосторонних связей* (multiway relationships), мы отложим до раздела 2.1.7 на с. 56.

2.1.4. Диаграммы сущностей и связей

Диаграмма сущностей и связей (entity–relationship diagram), или *ER-диаграмма* (ER-diagram), — это графическое представление *множеств сущностей*, их *атрибутов* и *связей*. Элементы названных видов описываются вершинами графа, и для задания принадлежности элемента к определенному виду используется специальная геометрическая фигура:

- *прямоугольник* — для множеств сущностей;
- *овал* — для атрибутов;
- *ромб* — для связей.

Ребра графа соединяют множества сущностей с атрибутами и служат для представления связей между множествами сущностей.

Пример 2.2. На рис. 2.2 приведена ER-диаграмма, представляющая структуру простой базы данных, содержащей информацию о кинофильмах. В составе диаграммы имеется три множества сущностей: *Movies* (“кинофильмы”), *Stars* (“актеры”) и *Studios* (“киностудии”).

Множество сущностей *Movies* обладает четырьмя атрибутами: *title* (“название”), *year* (“год производства”), *length* (“продолжительность”) и *filmType* (“тип пленки”) — “color” (“цветная”) или “blackAndWhite” (“черно-белая”). Два других множества сущностей, *Stars* и *Studios*, содержат по паре однотипных атрибутов, *name* (“имя” или “название”) и *address* (“адрес”), смысл которых вполне очевиден без дополнительных разъяснений. На диаграмме представлены две связи, описанные ниже.

1. *Stars-in* (“актеры-участники”, снявшиеся в фильме) — это связь, соединяющая каждую сущность-“кинофильм” с сущностями-“актерами”, принимавшими участие в съемках фильма. Связь *Stars-in*, рассматриваемая в противоположном направлении, в свою очередь соединяет актеров с кинофильмами.
2. Связь *Owns* (“владеет”) соединяет каждую сущность-“кинофильм” с сущностью-“студией”, выпустившей фильм и владеющей правами на него. Стрелка, задающая направление от связи *Owns* к множеству сущностей *Studios*, свидетельствует о том, что каждый фильм является собственностью одной и только одной киностудии. Подобные *ограничения уникальности* (uniqueness constraints) будут рассмотрены в разделе 2.1.6 на с. 55.

❑

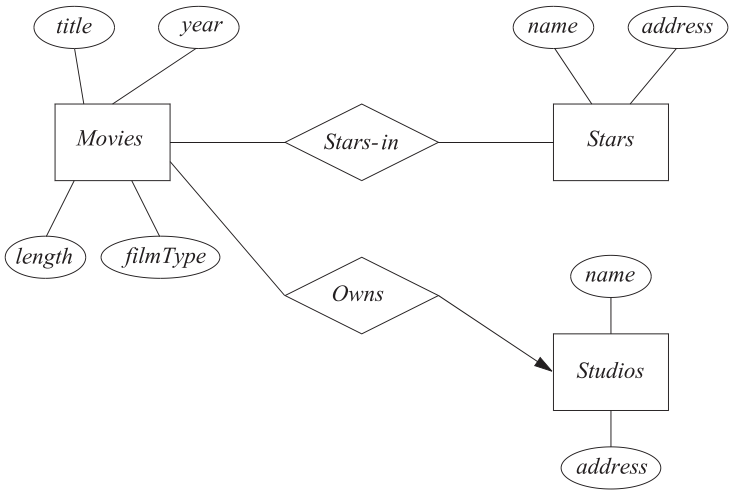

Рис. 2.2. Диаграмма сущностей и связей для базы данных о кинофильмах

2.1.5. Экземпляры ER-диаграммы

ER-диаграммы представляют собой инструмент описания *схем* (schemata), или структур, баз данных. Базу данных, соответствующую определенной ER-диаграмме и содержащую конкретный набор данных, принято называть *экземпляром* базы данных (database instance). Каждому множеству сущностей в экземпляре базы данных отвечает некоторый частный конечный набор сущностей, а каждая из таких сущностей обладает определенными значениями каждого из атрибутов. Уместно отметить, что информация о сущностях, атрибутах и связях носит строго абстрактный характер: содержимое ER-модели не может быть сохранено в базе данных непосредственно. Однако представление о том, что такие данные будто бы реально

существуют, помогает на начальной стадии проекта — пока мы не перейдем к отношениям и структуры данных не приобретут физическую форму.

Экземпляр базы данных включает также определенные экземпляры связей, описываемых диаграммой. Связи R, которая соединяет n множеств сущностей $E_1, E_2, ..., E_n$, соответствует экземпляр, состоящий из конечного множества списков $(e_1, e_2, ..., e_n)$, где каждый элемент e_i выбран из числа сущностей, присутствующих в текущем экземпляре множества сущностей E_i. Мы говорим, что элементы каждого из таких списков, охватывающих n сущностей, "соединены" посредством связи R.

Указанное множество списков называют *множеством данных связи* (relationship set) для текущего экземпляра связи R. Зачастую оказывается полезным представлять множество данных связи в виде таблицы. Столбцы этой таблицы озаглавлены наименованиями множеств сущностей, охватываемых связью, а каждому списку соединенных сущностей отводится одна строка таблицы.

Пример 2.3. Экземпляр связи *Stars-in* ("актеры-участники") легко описать таблицей пар данных, которая может иметь следующий вид:

Movies	*Stars*
`Basic Instinct`	`Sharon Stone`
`Total Recall`	`Arnold Schwarzenegger`
`Total Recall`	`Sharon Stone`

Члены множества данных связи — это строки таблицы. Например,

("Basic Instinct", "Sharon Stone")

представляет собой кортеж множества данных для текущего экземпляра связи *Stars-in*. ❑

2.1.6. Множественность бинарных связей

Бинарная связь (binary relationship) в общем случае способна соединять любой член одного множества сущностей с любым членом другого множества сущностей. Однако весьма распространены ситуации, в которых свойство "множественности" связи некоторым образом ограничивается. Предположим, что R — связь, соединяющая множества сущностей E и F. Тогда возможно выполнение одного из нескольких условий, перечисленных ниже.

- Если каждый член множества E посредством связи R может быть соединен не более чем с одним членом F, принято говорить, что R представляет связь типа *"многие к одному"* (many-one relationship), направленную от E к F. В этом случае каждая сущность множества F допускает соединение с многими членами E. Если же член F посредством связи R может быть соединен не более чем с одним членом E, мы говорим, что R — это связь "многие к одному", направленная от F к E (или, что то же самое, связь типа *"один ко многим"* (one-many relationship), направленная от E к F).
- Если связь R в обоих направлениях, от E к F и от F к E, относится к типу "мно-гие к одному", говорят, что R — это связь типа *"один к одному"* (one-one relationship). В этом случае каждый элемент одного множества сущностей допускает соединение не более чем с одним элементом другого множества сущностей.
- Если связь R ни в одном из направлений — ни от E к F и ни от F к E — не относится к типу "многие к одному", имеет место связь типа *"многие ко многим"* (many-many relationship).

Как мы уже отмечали в примере 2.2 (см. с. 53), стрелки в ER-диаграмме используются для отображения факта множественности связей. Если связь относится к типу "многие к одному"

и соединяет множество сущностей E с множеством сущностей F, она отображается в виде стрелки, направленной к F. Стрелка указывает, что каждая из сущностей множества E связана не более чем с одной сущностью множества F. Если при этом линия не снабжена противоположной стрелкой, обращенной к E, сущность множества F допускает связь со многими сущностями множества E.

Пример 2.4. Если следовать рассмотренной логике, связь типа "один к одному" между множествами сущностей E и F должна представляться на диаграмме двунаправленной стрелкой, один конец которой обращен в сторону множества E, а другой — в сторону F. На рис. 2.3 показаны два множества сущностей, *Studios* ("киностудии") и *Presidents* ("президенты"), соединенные связью *Runs* ("возглавляет") (атрибуты сущностей для краткости опущены). Уместно предположить, что каждый президент вправе руководить только одной студией, а каждая студия может возглавляться только одним президентом. Поэтому связь *Runs* следует отнести к типу "один к одному" и соединить на диаграмме с множествами сущностей *Studios* и *Presidents* посредством двух стрелок, по одной на каждое множество (так, как показано на рис. 2.3).

Рис. 2.3. Связь типа "один к одному"

Следует помнить: стрелка означает, что в связи участвует "не более чем один" элемент множества сущностей, на которое она указывает. При этом обязательное наличие такого элемента в составе множества не гарантируется. Рассматривая диаграмму рис. 2.3, мы вправе полагать, что некий "президент" обязательно связан с определенной студией — иначе на каком основании он мог бы величать себя президентом? Однако студия в какой-то период времени может обходиться без руководителя, так что стрелка, направленная от *Runs* к *Presidents*, на самом деле означает именно "не более чем один", но не "в точности один". Указанное различие мы проясним позже, в разделе 2.3.6 на с. 79. ❑

2.1.7. Многосторонние связи

ER-модели вполне по силам отображать связи, охватывающие *более* двух множеств сущностей. В реальных ситуациях *тернарные связи* (ternary relationships), соединяющие три множества, или связи, представляющие взаимоотношения еще большего числа множеств сущностей, сравнительно редки, но иногда они все-таки находят применение, помогая воссоздать в модели истинное положение вещей. *Многосторонние связи* (multiway relationships) отображаются на ER-диаграмме линиями, соединяющими ромб связи с каждым из соответствующих прямоугольников множеств сущностей.

Пример 2.5. На рис. 2.4 изображена связь *Contracts* ("контракты"), которая соединяет между собой множества сущностей *Studios* ("киностудии"), *Stars* ("актеры") и *Movies* ("кинофильмы"). Связь отображает факт заключения контракта между киностудией и определенным актером, обязующимся принять участие в съемках конкретного кинофильма. Значение некоторой связи в ER-модели, вообще говоря, можно воспринимать в виде соответствующего множества кортежей, компонентами которых являются

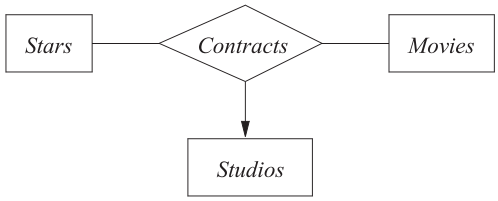

Рис. 2.4. Тернарная связь

Взаимоотношения типов связей

Следует отметить, что связь типа "многие к одному" является частным случаем связи типа "многие ко многим", а связь "один к одному" — это частный случай связи "многие к одному". Другими словами, любое свойство связей "многие ко многим" характерно и для связей "многие к одному", а некоторое свойство связей "многие к одному" сохраняется в силе для связей "один к одному". Например, структура данных, представляющая связь "многие к одному", способна адекватно отображать связи "один к одному", хотя в общем случае она непригодна для поддержки связей типа "многие ко многим".

сущности из множеств, соединяемых этой связью (мы говорили об этом в разделе 2.1.5 на с. 54). Таким образом, связь *Contracts* может быть описана набором кортежей вида

(studio, star, movie).

В многосторонних связях стрелка, обращенная к некоему множеству сущностей E, означает следующее: если мы выберем по одной сущности из всех остальных множеств сущностей, охватываемых связью, эти сущности могут быть связаны не более чем с одним элементом множества E. (Обратите внимание, что это правило является обобщением того, которое относится к бинарным связям типа "многие к одному".) На рис. 2.4 стрелка направлена к множеству сущностей *Studios*, свидетельствуя о том, что для каждой пары актеров и кинофильмов существует только одна студия, с которой этот актер заключил контракт на участие в съемках определенного кинофильма. Однако стрелки, которые были бы обращены к множествам сущностей *Stars* и *Movies*, не заданы: любая студия вправе пригласить для участия в фильме несколько актеров, а любой актер может быть связан со студией контрактом, предусматривающим участие в съемках нескольких кинофильмов. ❑

2.1.8. Связи и роли

Вполне вероятна ситуация, когда одно и то же множество сущностей упоминается в контексте единственной связи многократно. Если дело обстоит именно так, в ER-диаграмме задается столько линий, соединяющих связь с множеством сущностей, сколько требуется. Каждая линия, направленная к множеству сущностей, представляет отдельную *роль* (role), в которой множество выступает в конкретном случае. Линии, соединяющие связь и множество сущностей, принято обозначать текстовыми метками, описывающими определенные роли.

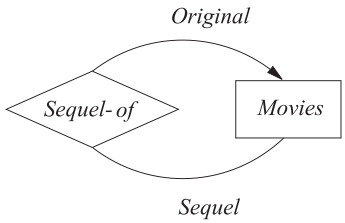

Рис. 2.5. Связь и ее роли

Пример 2.6. На рис. 2.5 изображена связь *Sequel-of* ("продолжение кинофильма"), соединяющая множество сущностей *Movies* ("кинофильмы") само с собой. Каждый конкретный экземпляр связи соединяет два кинофильма, один из ко-

торых служит продолжением другого. Чтобы различить два фильма, участвующих в связи, одна из ее линий помечена ролью *Original* (“исходный”), а другая — *Sequel* (“продолжение”). Мы подразумеваем, что некий фильм может иметь несколько продолжений, но для каждого продолжения существует только один “исходный” фильм. Таким образом, связь *Sequel-of*, соединяющая фильмы *Sequel* с фильмами *Original*, относится к типу “многие к одному” (этот факт на диаграмме рис. 2.5 отмечен стрелкой). ❑

Стрелки в многосторонних связях

Если связь охватывает три или более множеств сущностей, для адекватного описания каждой возможной ситуации средствами ER-диаграмм уже не достаточно ответить на вопрос, снабжать стрелкой соответствующую линию или нет. Для примера вновь обратимся к диаграмме рис. 2.4. Некоторая студия напрямую связана с определенным кинофильмом, а не с актером и кинофильмом, рассматриваемыми совместно, поскольку производством фильмов занимается именно студия. Однако используемая нами система обозначений не позволяет отличить эту ситуацию от случая, когда в тернарной связи одно множество сущностей в действительности является функцией двух других множеств. В разделе 3.4 на с. 106 мы рассмотрим строгую систему, основанную на задании функциональных зависимостей и позволяющую описать все мыслимые ситуации, в которых связи одного множества сущностей с другими обособлены друг от друга.

Пример 2.7. В качестве завершающего примера, иллюстрирующего как многосторонние связи, так и связи с несколькими ролями, на рис. 2.6 приведен более сложный вариант связи *Contracts* (“контракты”), рассмотренной выше, в примере 2.5 (см. с. 56). Теперь связь *Contracts* затрагивает уже две киностудии, актера и кинофильм. Смысл состоит в том, что одна студия, заключившая контракт с актером (вообще говоря, не обязательно связанный со съемками конкретного фильма), может подписать контракт с другой студией, который позволил бы актеру участвовать в работе над новым фильмом. Таким образом, связь теперь описывается набором кортежей следующего вида:

(studio1, studio2, star, movie).

Имеется в виду, что студия “studio2” заключает контракт со студией “studio1”, оговаривающий условия привлечения актера студии “studio1” на съемки фильма “movie”, который выпускается студией “studio2”.

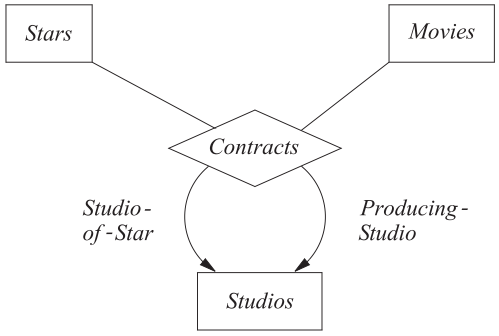

Рис. 2.6. Четырехсторонняя связь

Стрелки, изображенные на рис. 2.6, характеризуют две роли киностудии, относящейся к множеству сущностей *Studios*: “студия-владелец актера” (*Studio-of-Star*) и “студия-продюсер кинофильма” (*Producing-Studio*). Доводы таковы. Для каждого определенного актера, фильма и студии, которая занимается съемкой этого фильма, существует только одна студия, “владеющая” актером. (Предполагается, что актер заключил долговременный контракт только с одной студией.) Аналогично, конкретный фильм снимается только одной студией, так что обладая информацией об актере, фильме и студии, к которой относится этот актер, мы сможем определить уникальную сущность, соответствующую студии, осуществляющей съемку. Обратите внимание, что в обоих случаях для определения уникальной сущности нам необходимо только одно из остальных множеств сущностей — например, для отыскания

конкретной студии-продюсера достаточно определить кинофильм, снимаемый ею, — но этот факт не меняет общей картины множественности соединений в многосторонней связи.

Стрелок, которые были бы обращены к множествам *Movies* ("кинофильмы") и *Stars* ("актеры"), однако, не существует. Заданной тройке значений — имени актера и названиям студии-владельца и студии-продюсера — может соответствовать несколько контрактов, позволяющих актеру сниматься в различных фильмах. Поэтому такой набор данных кортежа не обязательно соответствует уникальному кинофильму. Аналогично, студия-продюсер вправе заключить контракт с другой студией на привлечение к съемкам фильма сразу нескольких актеров, так что имя актера в общем случае не может быть определено на основании данных трех других компонентов связи. ❑

2.1.9. Связи и атрибуты

Подчас бывает удобно или даже, как кажется, настоятельно необходимо ассоциировать атрибут со связью, а не с некоторым множеством сущностей, охватываемых этой связью. Вновь вернемся к примеру связи, показанной на рис. 2.4 (см. с. 56), которая представляет множество контрактов между актерами и студиями.[3] Пусть нам необходимо зафиксировать на диаграмме атрибут "размер заработной платы" (*salary*) актера (*Stars*), установленный в соответствии с контрактом (*Contracts*). Мы не вправе связывать подобный атрибут непосредственно с актером: последний за участие в съемках различных фильмов может получать различные суммы вознаграждения. Исходя из подобных соображений, не имеет смысла ассоциировать атрибут "размер заработной платы" и с множествами сущностей "киностудии" (*Studios*) (студии по-разному оплачивают работу различных актеров) и "кинофильмы" (*Movies*) (различные актеры за участие в съемках одного и того же фильма могут получать различную зарплату).

Однако уместно ассоциировать атрибут *salary* с кортежем

(star, movie, studio)

из множества данных, соответствующего связи *Contracts*. На рис. 2.7 приведена диаграмма рис. 2.4 (см. с. 56), дополненная атрибутами множеств сущностей *Movies*, *Stars* и *Studios* (эти атрибуты приводились на рис. 2.2 — см. с. 54), а также атрибутом *salary*, соединенным со связью *Contracts*.

[3] Здесь мы рассматриваем исходную, тернарную, редакцию связи *Contracts*, соответствующую примеру 2.5 на с. 56, а не ее четырехстороннюю версию, упоминавшуюся в примере 2.7 на с. 58.