Глава 1. Введение в системы баз данных

Понятие «система баз данных»

Система баз данных — это компьютеризированная система хранения записей. Саму же базу данных можно рассматривать как подобие электронной картотеки, т.е. хранилище или контейнер для некоторого набора занесенных в компьютер файлов данных. Пользователям этой системы предоставляется возможность выполнять множество различных операций над такими файлами, например:

- добавлять новые пустые файлы в базу данных;
- вставлять новые данные в существующие файлы;
- получать данные из существующих файлов;
- изменять данные в существующих файлах;
- удалять данные из существующих файлов;
- удалять существующие файлы из базы данных.

Основное назначение системы баз данных — хранить информацию, предоставляя пользователям средства её извлечения и модификации. К информации может относиться все, что заслуживает внимания отдельного пользователя или организации, использующей систему, иначе говоря, все необходимое для текущей работы данного пользователя или предприятия.

На рисунке 1.1 показана упрощённая схема системы баз данных. Здесь отражено четыре главных компонента системы:

1. данные;
2. аппаратное обеспечение;
3. программное обеспечение;
4. пользователи.

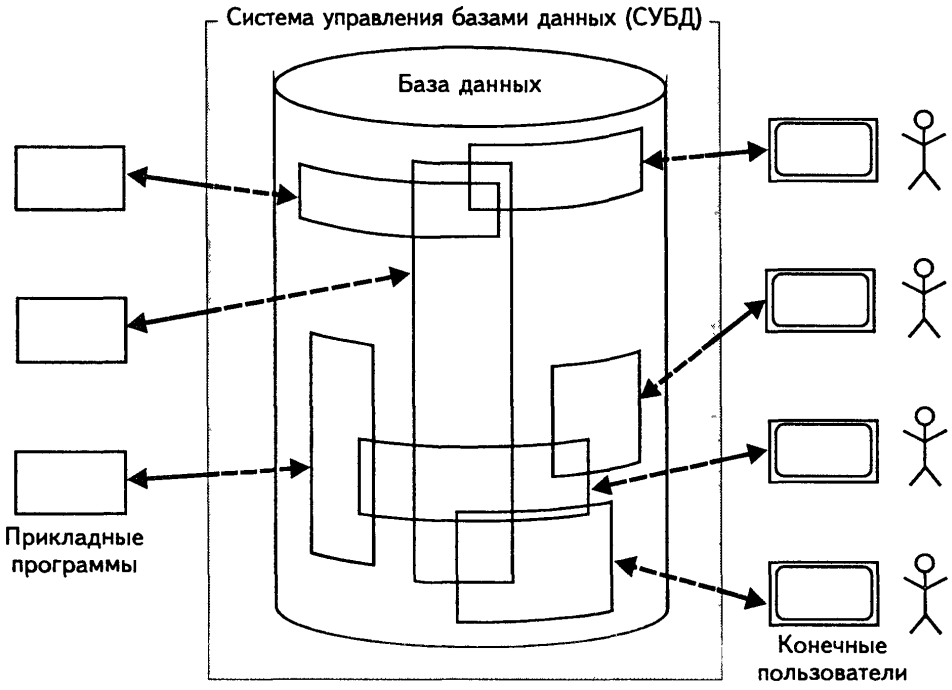

Рисунок 1.1. Упрощённая схема системы баз данных

Данные. Системы с базами данных существуют как на самых малых компьютерах, так и на крупнейших мэйнфреймах. Нет необходимости говорить, что предоставляемые каждой конкретной системой средства в некоторой мере зависят от мощности и возможностей базовой машины. В частности, системы на больших машинах («большие системы»), в основном, многопользовательские, тогда как системы на малых машинах («малые системы»), как правило, однопользовательские. Однопользовательская система (single-user system) — это система, в которой одновременно к базе данных может получить доступ не более одного пользователя, а многопользовательская система (multi-user system) — это такая система, в которой к базе данных могут получить доступ сразу несколько пользователей. Как и в схеме на рисунке 1.1, мы обычно будем подразумевать именно второй вид систем, хотя, с точки зрения пользователей, между этими системами фактически не существует большого различия. Основная задача большинства многопользовательских систем — позволить каждому отдельному пользователю работать с ней так, как он мог бы работать с однопользовательской системой. Различия между этими двумя видами систем проявляются в их внутренней структуре, и потому практически не видны конечному пользователю.

В общем случае данные в базе данных являются *интегрированными* и *разделяемыми*.

Под понятием ***интегрированности*** данных подразумевается возможность представить базу данных как объединение нескольких отдельных файлов данных, полностью или частично исключающее избыточность хранения информации.

Под понятием ***разделяемости*** данных подразумевается возможность использования отдельных элементов, хранимых в базе данных, несколькими различными пользователями. Имеется в виду, что каждый из пользователей сможет получить доступ к одному и тому же элементу данных, возможно, для достижения различных целей. Кроме того, разные пользователи могут получать доступ к одному и тому же элементу данных в одно и то же время (параллельный доступ). Если база данных не является разделяемой, то ее иногда называют личной или базой данных специального назначения.

Одним из следствий этих двух характеристик базы данных является то, что каждый конкретный пользователь обычно имеет дело лишь с небольшой частью всей базы данных, причём обрабатываемые различными пользователями части могут произвольным образом перекрываться. Иначе говоря, каждая база данных воспринимается её различными пользователями по-разному. Фактически даже те два пользователя базы данных, которые работают с одними и теми же элементами данных, могут иметь значительно отличающиеся представления о них.

Аппаратное обеспечение. К аппаратному обеспечению системы относятся:

- тома вторичной (внешней) памяти (обычно это магнитные диски), используемые для хранения информации, а также соответствующие устройства ввода-вывода (дисководы и т.п.), контроллеры устройств, каналы ввода-вывода и т.д.
- аппаратный процессор (или процессоры) вместе с основной (первичной) памятью, предназначенные для поддержки работы программного обеспечения системы баз данных.

В целом, аппаратное обеспечение представляет собой обычный персональный компьютер, который используется в качестве *сервера баз данных*. Кроме самой системы баз данных сервер может содержать приложения, которые взаимодействуют с базами данных, а также предоставлять доступ к системе баз данных другим устройствам в сети.

Программное обеспечение. Между собственно физической базой данных (т.е. данными, которые реально хранятся) и пользователями системы располагается уровень программного обеспечения, который можно называть по-разному: *менеджер базы данных* (*database manager*), *сервер базы данных* (*database server*) или, что более привычно, *система управления базами данных, СУБД* (*database management system — DBMS*). Все запросы пользователей на доступ к базе данных обрабатываются СУБД. Все имеющиеся средства добавления файлов (или таблиц), выборки и обновления данных в этих файлах или таблицах также предоставляет СУБД. Основная задача СУБД — предоставить пользователю базы данных возможность работать с ней, *не вникая в детали на уровне аппаратного обеспечения*. Иными словами, СУБД позволяет конечному пользователю рассматривать базу данных как объект более высокого уровня по сравнению с аппаратным обеспечением, а также предоставляет в его распоряжение набор операций, выражаемых в терминах языка высокого уровня.

СУБД — это наиболее важный, но не единственный программный компонент системы. Среди других компонентов — утилиты, средства разработки приложений, средства проектирования, генераторы отчетов и *менеджер транзакций* (*transaction manager*) или *диспетчер выполнения транзакций* (*TP monitor*).

Термин «СУБД» также часто используется в отношении конкретных программных продуктов конкретных изготовителей. Иногда, в тех случаях, когда конкретная копия подобного продукта устанавливается для работы на определённом компьютере, используется термин *экземпляр СУБД*. Эти два понятия необходимо чётко различать.

Следует иметь в виду, что в среде изготовителей термин «база данных» часто используется даже тогда, когда на самом деле подразумевается СУБД, что влечёт за собой неоднозначность и путаницу.

Пользователи. Пользователей можно разделить на три большие и отчасти перекрывающиеся группы:

- ***прикладные программисты*** — отвечают за написание прикладных программ, использующих базу данных. Для этих целей применим практически любой современный язык программирования высокого уровня. Прикладные программы получают доступ к базе данных посредством выдачи соответствующего запроса к СУБД (обычно это некоторый SQL-оператор). Подобные программы могут быть простыми пакетными приложениями или же интерактивными приложениями, предназначенными для поддержки работы конечных пользователей. В последнем случае они предоставляют пользователям непосредственный оперативный доступ к базе данных через рабочую станцию или терминал. Большинство современных приложений относится именно к этой категории;
- ***конечные пользователи*** — работают с системой баз данных непосредственно через рабочую станцию или терминал. Конечный пользователь может получать доступ к базе данных, применяя одно из интерактивных приложений, упомянутых выше, или же интерфейс, интегрированный в программное обеспечение самой СУБД. Безусловно, подобный интерфейс также поддерживается интерактивными приложениями, однако эти приложения не создаются пользователями-программистами, а являются встроенными в СУБД. Большинство СУБД включает, по крайней мере, одно такое встроенное приложение, а именно — *процессор языка запросов*, позволяющий пользователю в диалоговом режиме вводить запросы к базе данных. Запросы часто называют *операторами* (*statement*) или *командами* (*commands*).

Кроме языка запросов, в большинстве систем дополнительно предоставляются специализированные встроенные интерфейсы, в которых пользователь в явном виде не использует команд, подобных оператору SELECT. Работа с базой данных осуществляется за счёт выбора пользователем необходимых команд меню или заполнения требуемых полей в предоставленных формах. Такие *некомандные* интерфейсы, основанные на меню и формах, облегчают работу с базами данных тех, кто не имеет опыта работы с информационными системами. *Командный* интерфейс, т.е. язык запросов, напротив, требует некоторого профессионального опыта работы с информационными системами, однако командный интерфейс более гибок, чем некомандный. К тому же языки запросов обычно включают определённые функции, отсутствующие в некомандных интерфейсах.

- ***администраторы базы данных*** — занимаются решением различных задач, связанных с поддержанием работоспособности баз данных.

Понятие «база данных»

База данных — это некоторый набор перманентных (постоянных) данных, используемых прикладными системами какого-либо предприятия.

Под словом *перманентные* (*persistent*) подразумеваются данные, которые отличаются от других, более изменчивых данных, таких как промежуточные результаты, входные и выходные данные, управляющие операторы, рабочие очереди, программные управляющие блоки и вообще все *временные* (*transient*) по своей сути данные. Точнее говоря, можно утверждать, что данные в базе остаются «перманентными», поскольку после того, как они были приняты средствами СУБД для помещения в базу, удалить их из неё впоследствии можно лишь с помощью соответствующего явного запроса к базе данных, но не как результат какого-либо побочного эффекта от выполнения некоторой программы.

Здесь слово «предприятие» — это общий термин для относительно независимой коммерческой, научной, технической или любой другой организации или предприятия. Предприятие может состоять всего из одного человека (с небольшой частной базой данных), быть целой корпорацией или другой крупной организацией (с очень большой общей базой данных) либо представлять собой нечто среднее между этими крайними случаями. Любое предприятие неизбежно использует большое количество данных, связанных с его деятельностью. Это и есть «перманентные данные».

Обычно база данных содержит совокупность *сущностей*, соединённых друг с другом с помощью *связей*. Термин «сущность» обычно используется в теории баз данных для обозначения любого отличимого объекта, который может быть представлен в базе данных. Связи между сущностями обычно являются двусторонними, т.е. их можно рассматривать в любом направлении.

Сущность — это то, о чём необходимо записывать информацию. Отсюда следует, что сущности и связи имеют некоторые *свойства* (*properties*), соответствующие тем данным о них, которые необходимо сохранить в базе данных. В общем случае свойства могут быть как простыми, так и сложными, причем настолько, насколько это потребуется. Однако, чаще всего свойства являются простыми, и их можно представить простыми типами данных: числами, строками, датами, отметками времени и т.п.

Существует и другой, не менее важный, подход к пониманию данных и баз данных. Слово «данные» (*data*) происходит от латинского слова «давать», откуда следует, что данные на самом деле являются заданными *фактами*, из которых можно логически получить другие факты. (Получение

дополнительных фактов из заданных фактов — это в точности то, для чего используется СУБД, обслуживая запросы пользователя.) «Заданный факт», в свою очередь, соответствует тому, что в логике называется истинным высказыванием. Отсюда следует, что база данных в действительности есть набор подобных истинных высказываний.

Подобная интерпретация данных легла в основу реляционных систем баз данных. Реляционные системы основаны на формальной теории, называемой *реляционной моделью данных*, которая предполагает следующее.

Данные представлены посредством строк в таблицах, и эти строки могут быть непосредственно интерпретированы как истинные высказывания. Для обработки строк данных предоставляются операторы, которые напрямую поддерживают процесс логического получения дополнительных истинных высказываний из существующих высказываний.

Однако реляционная модель — не единственная возможная модель данных. Существуют и другие модели, хотя многие из них отличаются от реляционной модели только тем, что они в определённой степени приспособлены для специальных случаев, а не строго построены на формальной логике.

Модель данных — это абстрактное, самодостаточное, логическое определение объектов, операторов и прочих элементов, в совокупности составляющих абстрактную машину, с которой взаимодействует пользователь. Упомянутые объекты позволяют моделировать структуру данных, а операторы — поведение данных.

Используя это определение, можно эффективно разделить модель данных и её реализацию.

Реализация (*implementation*) заданной модели данных — это физическое воплощение на реальной машине компонентов абстрактной машины, которые в совокупности составляют эту модель.

Проще говоря, модель — это то, что пользователи должны знать, а реализация — это то, чего пользователи знать не должны.

Назначение баз данных

Почему используются системы с базами данных? Какие преимущества получает пользователь при работе с ними? В некоторой степени ответ зависит от того, о какой системе идет речь — однопользовательской или многопользовательской (точнее будет сказать, что существуют многочисленные *дополнительные* преимущества использования многопользовательских систем).

В случае однопользовательской системы базы данных предоставляют ряд преимуществ по сравнению с методом «ручного» учёта данных:

- *компактность* — нет необходимости в создании и ведении многотомных бумажных картотек;
- *скорость* — компьютер может выбирать и обновлять данные гораздо быстрее человека. В частности, с его помощью можно быстро получать ответы на произвольные вопросы, возникающие в процессе работы, не затрачивая времени на визуальный поиск или поиск вручную;
- *низкие трудозатраты* — нет необходимости в утомительной работе над картотекой вручную. Механическую работу машины всегда выполняют лучше;

- *актуальность* — в случае необходимости под рукой в любой момент имеется точная свежая информация.

Эти преимущества приобретают ещё большее значение в многопользовательской среде, где база данных, вероятно, больше и сложнее однопользовательской. Кроме того, многопользовательская среда имеет дополнительное преимущество: система баз данных предоставляет предприятию средства централизованного управления его данными (именно возможность такого управления является наиболее ценным свойством базы данных). Представьте себе противоположную ситуацию: предприятие не использует систему баз данных, в которой для каждого отдельного приложения создаются свои файлы, чаще всего размещаемые на отдельных магнитных лентах или дисках, в результате чего данные оказываются разрозненными. Систематически управлять такими данными очень сложно.

Рассмотрим более подробно концепцию централизованного управления. Предполагается, что при централизованном управлении на предприятии, использующем базу данных, есть человек, который несёт основную ответственность за данные предприятия. Это *администратор данных*. В связи с тем, что данные — это одна из главных ценностей предприятия, администратор должен разбираться в них и понимать нужды предприятия по отношению к данным на уровне высшего управляющего звена в руководстве предприятием. Сам администратор данных также должен относиться к этому звену. В его обязанности входит принятие решений о том, какие данные необходимо вносить в базу данных в первую очередь, а также выработка требований по сопровождению и обработке данных после их занесения в базу данных. Примером подобных требований может служить распоряжение о том, кто и при каких обстоятельствах имеет право выполнять конкретные операции над теми или иными данными. Другими словами, администратор данных должен обеспечивать *защиту данных.*

Очень важно помнить, что администратор данных относится к управляющему звену, а не к техническим специалистам (хотя он, конечно, должен иметь хорошее представление о возможностях баз данных на техническом уровне). Технический специалист, ответственный за реализацию решений администратора данных, — это *администратор базы данных*. В отличие от администратора данных, администратор базы данных должен быть профессиональным специалистом в области информационных технологий. Его работа заключается в создании самих баз данных и организации технического контроля, необходимого для осуществления решений, принятых администратором данных. Администратор базы данных также несёт ответственность за обеспечение необходимого быстродействия системы и её техническое обслуживание. Обычно у администратора базы данных есть штат из системных программистов и технических ассистентов. Однако для простоты удобнее считать, что администратор базы данных — один человек.

Возможность централизованного управления базами данных предоставляет следующие преимущества.

Возможность совместного доступа к данным. Совместный доступ к данным означает не только возможность доступа к ним нескольких существующих приложений базы данных, но и возможность разработки новых приложений для работы с этими же данными. Другими словами, требования новых приложений по доступу к данным могут быть удовлетворены без необходимости добавления новых данных в базу.

Сокращение избыточности данных. В системах, не использующих базы данных, каждое приложение имеет свои файлы. Это часто приводит к избыточности хранимых данных и, следовательно,

к расточительству пространства вторичной памяти. Например, как приложение, связанное с учётом персонала, так и приложение, связанное с учетом обучения служащих, могут иметь собственные файлы с ведомственной информацией о служащих. Эти два файла можно объединить с устранением избыточности (одинаковой информации) при условии, что администратор данных знает о том, какие данные нужны для каждого приложения, т.е. что на предприятии осуществляется необходимое общее управление. В данном случае мы не имеем в виду, что избыточность данных может или должна быть устранена *полностью*. Иногда веские практические или технические причины требуют наличия нескольких копий хранимых данных. Однако такая избыточность должна строго контролироваться, т.е. учитываться в СУБД. Кроме того, в подобном случае должна быть предусмотрена возможность «распространения обновлений».

Устранение противоречивости данных. В действительности это следствие предыдущего пункта. Пусть служащий с номером 'E3', работающий в отделе с номером 'D8', представлен двумя различными записями в базе данных. Предположим, что в СУБД не учтено это раздвоение (т.е. избыточность данных не контролируется). Тогда рано или поздно обязательно возникнет ситуация, при которой эти две записи перестанут быть согласованными, когда одна из них будет изменена, а другая — нет. В этом случае база данных станет *противоречивой*. Ясно, что противоречивая база данных способна предоставлять пользователю неправильную, противоречивую информацию. Также очевидно, что если какой-либо факт представлен одной записью (т.е. при отсутствии избыточности), то противоречий возникнуть не может. Противоречий можно также избежать, если не удалять избыточность, а *контролировать* её (соответствующим образом известив об этом СУБД). Тогда СУБД сможет гарантировать, что, с точки зрения пользователя, база данных никогда не будет противоречивой. Данная гарантия обеспечивается тем, что если обновление вносится в одну запись, то оно автоматически будет распространено на все остальные. Этот процесс называется *распространением обновлений* (*propagating updates*).

Возможность поддержки транзакций. ***Транзакция*** (*transaction*) — это логическая единица работы, обычно включающая несколько операций базы данных (в частности, несколько операций изменения). Стандартный пример — передача суммы денег со счёта A на счёт B. Очевидно, что в данном случае необходимы два изменения: изъятие денег со счёта A и их внесение на счёт B. Если пользователь укажет, что оба изменения входят в одну и ту же транзакцию, то система сможет реально гарантировать, что либо оба эти изменения будут выполнены, либо не будет выполнено ни одно из них, даже если до завершения процесса изменений в системе произойдет сбой (скажем, из-за перерыва в подаче электроэнергии).

Обеспечение целостности данных. Задача обеспечения целостности заключается в гарантированной поддержке корректности данных в базе. Противоречивость между двумя записями, представляющими один «факт», является примером утраты целостности данных. Конечно, эта конкретная проблема может возникнуть лишь при наличии избыточности в хранимых данных. Но даже если избыточность отсутствует, база данных может содержать некорректную информацию. Централизованное управление базой данных позволяет избежать подобных проблем (насколько их вообще возможно избежать). Для этого администратор данных определяет (а администратор базы данных реализует) *ограничения целостности* (*integrity constraints*), иначе называемые *бизнес-правилами*, которые будут применяться при любой попытке внести какие-либо изменения в соответствующие данные. Целостность данных для многопользовательских систем баз данных даже более важна, чем для среды с «частными файлами», причём именно по той причине, что такая база данных поддерживает совместный доступ. При отсутствии должного контроля один пользователь вполне может некорректно обновить данные, от чего пострадают многие другие ни в чём не повинные пользователи.

Организация защиты данных. Благодаря полному контролю над базой данных АБД (безусловно, в соответствии с указаниями администратора данных) может обеспечить доступ к ней только через определенные каналы. Для этой цели могут устанавливаться *ограничения защиты* (*security constraints*) или правила, которые будут проверяться при любой попытке доступа к уязвимым данным. Можно установить различные правила для разных типов доступа (выборка, вставка, удаление и т.д.) к каждому из элементов информации в базе данных. Однако следует заметить, что при отсутствии таких правил безопасность данных подвергается большему риску, чем в обычной (разобщенной) файловой системе. Следовательно, централизованная природа системы баз данных в определённом смысле требует наличия надежной системы защиты.

Возможность балансировки противоречивых требований. Зная общие требования всего предприятия (а не требования каждого отдельного пользователя), администратор базы данных (опять же, в соответствии с указаниями администратора данных) может структурировать базу данных таким образом, чтобы обслуживание было наилучшим для всего предприятия. Например, он может выбрать такое физическое представление данных во вторичной памяти, которое обеспечит быстрый доступ к информации для наиболее важных приложений (возможно, с потерей производительности для некоторых других приложений).

Возможность введения стандартизации. Благодаря централизованному управлению базой данных администратор базы данных (по указаниям администратора данных) может обеспечить соблюдение всех подходящих стандартов, регламентирующих представление данных в системе. Стандарты могут быть частными, корпоративными, ведомственными, промышленными, национальными и интернациональными. Стандартизация представления данных наиболее важна с точки зрения обмена и пересылки данных между системами. Кроме того, стандарты именования и документирования данных важны как в отношении их совместного использования, так и в отношении их углубленного понимания.

Обеспечение независимости от данных. Обеспечение независимости данных — важнейшая цель создания систем баз данных. Одна и та же информация может по-разному представляться в различных приложениях, работающих с одной и той же базой данных. При этом, требуется, чтобы изменения данных этими приложениями корректно обрабатывались СУБД и приводились в тому виду, в котором эти данные хранятся непосредственно в базе данных. При этом, формат хранимых данных может меняться, но эти изменения не должны приводить к необходимости изменять работающие с этими данными приложения. Независимость данных можно определить как иммунитет приложений к изменениям в физическом представлении данных и в методах доступа к ним, а это означает, что рассматриваемые приложения не зависят от любых конкретных способов физического представления информации или выбранных методов доступа к ним.

Системы управления базами данных

В настоящее время на рынке баз данных преобладают СУБД, основанные на *реляционной модели данных*. Реляционная модель основана на определённых математических и логических принципах и, следовательно, идеально подходит для изложения теоретических концепций систем баз данных.

Реляционная система основывается на следующих принципах.

1. Данные передаются пользователю в виде таблиц, и никак иначе.

2. Пользователю предоставляются операторы (например, для выборки данных), позволяющие генерировать новые таблицы на основании уже существующих. Например, в системе обязательно должны присутствовать оператор *ограничения*, предназначенный для получения подмножества строк заданной таблицы, и оператор *проекции*, позволяющий получить подмножество её столбцов. Однако подмножество строк и подмножество столбцов некоторой таблицы, безусловно, можно рассматривать как новые таблицы.

На практике системы баз данных могут быть легко распределены по категориям в соответствии со структурами данных и операторами, которые они предоставляют пользователю. Прежде всего, старые (дореляционные) системы можно разделить на три большие категории:

- системы инвертированных списков (*inverted list*);
- иерархические (*hierarchic*);
- сетевые (*network*).

Реляционные и нереляционные системы можно различать по следующим признакам. Как уже отмечалось, пользователь реляционной системы видит данные в виде таблиц и никак иначе. Пользователь нереляционной системы, напротив, видит данные, представленные в других структурах: либо вместо таблиц реляционной системы, либо наряду с ними. Для работы с этими другими структурами применяются другие операции. Например, в иерархической системе данные представляются пользователю в форме набора древовидных структур (иерархий), а среди операций работы с иерархическими структурами есть операции перемещения по иерархическим указателям (навигации) вверх и вниз по ветвям деревьев. Сетевые системы, подобно иерархическим, предоставляют в распоряжение пользователя внутренние указатели на элементы данных. Реляционные системы не имеют таких указателей, и это очень важная их отличительная особенность.

Первые реляционные СУБД начали появляться в конце 1970-х и начале 1980-х годов. На сегодняшний день существует довольно большое количество СУБД, которые работают на практически любой программной и аппаратной компьютерной платформе. Мы будем рассматривать на практике СУБД трёх производителей: Microsoft SQL Server Express, PostgreSQL и MySQL. Эти СУБД являются бесплатными, и в то же время обладают достаточной функциональностью для создания и работы с базами данных, и их можно использовать как часть программного комплекса для решения каких-либо задач, связанных с хранением и обработкой данных.

Microsoft SQL Server выпускается компанией Microsoft с конца 1980-х годов и позиционируется как система анализа и управления реляционными базами данных в решениях электронной коммерции. Последняя на текущий момент версия — SQL Server 2017, которая выпускается в нескольких редакциях:

- *Express* — это бесплатная редакция системы SQL Server. Она предназначается для обучения работе с базами данных, для создания небольших серверных приложений и для распространения независимыми поставщиками ПО;
- *Standard* — эта редакция предназначена для управления данными и выполнения операций бизнес-аналитики. Она демонстрирует лучшие в своём классе показатели простоты использования и степени управляемости приложений, на которых основана работа подразделений предприятия;
- *Enterprise* — это комплексная платформа управления данными и бизнес-аналитики, предназначена для критически важных приложений и больших хранилищ данных. Она обладает

первоклассной масштабируемостью, возможностью создавать хранилища данных, продвинутыми средствами анализа и достаточной безопасностью, что позволяет использовать её как основу для критически важных бизнес-приложений.

В версии SQL Server Express присутствуют ограничения по использованию аппаратных ресурсов компьютера: она использует только 4 ядра процессора, и размер базы данных не может превышать 10 Гб. Эти ограничения несущественны для небольших баз данных, но в случае более масштабного использования необходимо использовать одну из платных версий.

PostgreSQL — это свободная объектно-реляционная система управления базами данных. Существует в реализациях для множества UNIX-платформ, а также для Windows. Сильными сторонами PostgreSQL считаются:

- поддержка БД практически неограниченного размера;
- мощные и надёжные механизмы транзакций и репликации;
- расширяемая система встроенных языков программирования: в стандартной поставке поддерживаются PL/pgSQL, PL/Perl, PL/Python и PL/Tcl; дополнительно можно использовать PL/Java, PL/PHP, PL/Py, PL/R, PL/Ruby, PL/Scheme и PL/sh, а также имеется поддержка загрузки C-совместимых модулей;
- наследование;
- лёгкая расширяемость.

PostgreSQL является полностью бесплатной, и по функциональности схожа с Microsoft SQL Server Express, но не имеет ограничений по использованию аппаратных средств компьютера. Также существуют более мощные варианты этой СУБД, выпущенные компанией EnterpriseDB — Postgres Plus и Postgres Plus Advanced Server. Они включают большой набор программного обеспечения для разработчиков и анализа данных и являются платными для коммерческого использования.

MySQL — это свободная система управления базами данных. Разработку и поддержку MySQL осуществляет корпорация Oracle, получившая права на торговую марку вместе с поглощённой Sun Microsystems, которая ранее приобрела шведскую компанию MySQL AB. MySQL может распространяться бесплатно в соответствии с условиями лицензии GPL. Однако по условиям GPL, если какая-либо программа включает исходные коды MySQL, то она тоже должна распространяться по лицензии GPL. Это может расходиться с планами разработчиков, не желающих открывать исходные тексты своих программ. Для таких случаев предусмотрена коммерческая лицензия.

MySQL является решением для малых и средних приложений. Обычно MySQL используется в качестве сервера, к которому обращаются локальные или удалённые клиенты, однако в дистрибутив входит библиотека внутреннего сервера, позволяющая включать MySQL в автономные программы. Базы данных MySQL можно использовать совместно с большим количеством языков и сред программирования, что обусловило широкое распространение этой СУБД.

Краткие итоги

1. Систему баз данных можно рассматривать как компьютеризированную систему хранения записей. Такая система включает сами по себе данные (сохраняемые в базе данных), аппаратное обеспечение, программное обеспечение (в частности, систему управления базами данных, или СУБД), а также пользователей (что наиболее важно).

2. Пользователи, в свою очередь, подразделяются на прикладных программистов, конечных пользователей и администраторов баз данных, или АБД. Последние отвечают за администрирование базы данных и всей системы баз данных в соответствии с требованиями, устанавливаемыми администратором данных.
3. Базы данных являются интегрированными и чаще всего совместно используемыми. Они применяются для хранения перманентных данных. Можно считать, что эти данные представляют собой сущности и соединяются друг с другом с помощью связей.
4. Система баз данных имеет ряд преимуществ, наиболее важным из которых является физическая независимость данных. Независимость данных может быть определена как иммунитет прикладных программ к изменениям способа хранения данных и используемых методов доступа.
5. Система баз данных может быть основана на нескольких различных подходах. Реляционные системы базируются на формальной теории, называемой реляционной моделью, в соответствии с которой данные представляются в виде строк в таблицах (и интерпретируются как истинные высказывания), а пользователям предоставляются операторы, обеспечивающие поддержку процесса получения дополнительных истинных высказываний в виде следствий из существующих данных.
6. Большинство современных СУБД являются реляционными. Существуют несколько бесплатных версий СУБД, которые обладают схожей функциональностью и могут использоваться в разрабатываемых вами приложениях.

Глава 2. Модели данных

Уровни и типы моделей данных

СУБД должна предоставлять доступ к данным любым пользователям, включая и тех, которые практически не имеют и (или) не хотят иметь представления о физическом размещении в памяти данных и их описаний, механизмах поиска запрашиваемых данных, проблемах, возникающих при одновременном запросе одних и тех же данных многими пользователями, способах обеспечения защиты данных от некорректных обновлений и несанкционированного доступа, поддержании баз данных в актуальном состоянии и множестве других функций СУБД.

При выполнении основных из этих функций СУБД должна использовать различные описания данных, которые сначала необходимо создать.

Естественно, что проект базы данных надо начинать с анализа предметной области и выявления требований к ней отдельных пользователей, например, сотрудников организации, для которых создаётся база данных. Обычно проектирование структуры базы данных поручается *администратору базы данных*. Им может быть как специально выделенный сотрудник организации, так и будущий пользователь базы данных, достаточно хорошо знакомый с машинной обработкой данных.

Объединяя частные представления о содержимом базы данных, полученные в результате опроса пользователей, и свои представления о данных, которые могут потребоваться в будущих приложениях, администратор базы данных сначала создает обобщенное неформальное описание создаваемой базы данных. Это описание, выполненное с использованием естественного языка, математических формул, таблиц, графиков и других средств, понятных всем людям, работающих над проектированием базы данных, называют *инфологической моделью данных* (рисунок 2.1).

Рисунок 2.1. Уровни моделей данных

Такая человеко-ориентированная модель полностью независима от физических параметров среды хранения данных. Инфологическая модель не должна изменяться до тех пор, пока какие-то изменения в реальном мире не потребуют изменения в ней некоторого определения, чтобы эта модель продолжала отражать предметную область.

Остальные модели, показанные на рисунке 2.1, являются компьютеро-ориентированными. С их помощью СУБД даёт возможность программам и пользователям осуществлять доступ к хранимым данным лишь по их именам, не заботясь о физическом расположении этих данных. Нужные данные отыскиваются СУБД на внешних запоминающих устройствах по *физической модели данных*.

Так как указанный доступ осуществляется с помощью конкретной СУБД, то модели должны быть описаны на *языке описания данных* этой СУБД. Такое описание, создаваемое администратором базы данных по инфологической модели данных, называют *даталогической моделью данных*.

Трёхуровневая архитектура СУБД (инфологический, даталогический и физический уровни) позволяет обеспечить *независимость хранимых данных* от использующих их программ. Администратор базы данных может при необходимости переписать хранимые данные на другие носители информации или реорганизовать их физическую структуру, изменив лишь физическую модель данных. Также администратор базы данных может подключить к системе любое число новых пользователей (новых приложений), дополнив при необходимости даталогическую модель. Указанные изменения физической и даталогической моделей не будут замечены существующими пользователями системы (окажутся «прозрачными» для них), так же как не будут замечены и новые пользователи. Следовательно, независимость данных обеспечивает возможность развития системы баз данных без разрушения существующих приложений.

Инфологическая модель отображает реальный мир в некоторые понятные человеку концепции, полностью независимые от параметров среды хранения данных. Существует множество подходов к построению таких моделей: графовые модели, семантические сети, модель «сущность—связь» и др. Наиболее популярной из них является модель данных «сущность—связь».

Инфологическая модель должна быть отображена в компьютеро-ориентированную даталогическую модель, «понятную» СУБД. В процессе развития теории и практического использования баз данных, а также средств вычислительной техники создавались СУБД, поддерживающие различные даталогические модели.

Сначала стали использовать иерархические даталогические модели. Простота организации, наличие заранее заданных связей между сущностями, сходство с физическими моделями данных позволяли добиваться приемлемой производительности иерархических СУБД на медленных ЭВМ с весьма ограниченными объемами памяти. Но, если данные не имели древовидной структуры, то возникала масса сложностей при построении иерархической модели и желании добиться нужной производительности.

Сетевые модели также создавались для мало ресурсных ЭВМ. Это достаточно сложные структуры, состоящие из «наборов» — поименованных двухуровневых деревьев. «Наборы» соединяются с помощью «записей-связок», образуя цепочки, и т.д. При разработке сетевых моделей было придумано множество «маленьких хитростей», позволяющих увеличить производительность СУБД, но существенно усложнивших последние. Прикладной программист должен знать массу терминов, изучить несколько внутренних языков СУБД, детально представлять логическую структуру базы данных для осуществления навигации среди различных экземпляров, наборов, записей и т.п.

Сложность практического использования иерархических и сетевых СУБД заставляла искать иные способы представления данных. В конце 60-х годов появились СУБД на основе инвертированных файлов, отличающиеся простотой организации и наличием весьма удобных языков манипулирования данными. Однако такие СУБД обладают рядом ограничений на количество файлов для хранения данных, количество связей между ними, длину записи и количество её полей. На сегодняшний день наиболее распространены реляционные модели.

Модель данных «сущность—связь»

Цель инфологического моделирования — обеспечение наиболее естественных для человека способов сбора и представления той информации, которую предполагается хранить в создаваемой базе данных. Поэтому инфологическую модель данных пытаются строить по аналогии с естественным языком (последний не может быть использован в чистом виде из-за сложности компьютерной обработки текстов и неоднозначности любого естественного языка). Основными конструктивными элементами инфологических моделей являются *сущности*, *связи* между ними и их *свойства*, или *атрибуты*.

Сущность — это любой различимый объект (объект, который можно отличить от другого объекта), информацию о котором необходимо хранить в базе данных. Сущностями могут быть люди, места, самолеты, рейсы, вкус, цвет и т.д. Необходимо различать такие понятия, как *тип сущности* и *экземпляр сущности*. Понятие тип сущности относится к набору однородных личностей, предметов, событий или идей, выступающих как целое. Экземпляр сущности относится к конкретной вещи в наборе. Например, типом сущности может быть ГОРОД, а экземпляром — Москва, Киев и т.д.

Атрибут — это поименованная характеристика сущности. Его наименование должно быть уникальным для конкретного типа сущности, но может быть одинаковым для различного типа сущностей (например, ЦВЕТ может быть определён для многих сущностей: СОБАКА, АВТОМОБИЛЬ, ДОМ и т.д.). Атрибуты используются для определения того, какая информация должна быть собрана о сущности. Примерами атрибутов для сущности АВТОМОБИЛЬ являются ТИП, МАРКА, НОМЕРНОЙ ЗНАК, ЦВЕТ и т.д. Здесь также существует различие между типом и экземпляром. Тип атрибута ЦВЕТ имеет много экземпляров или значений: Красный, Синий, Банановый, Белая ночь и т.д. Однако каждому экземпляру сущности присваивается только одно значение атрибута.

Абсолютное различие между типами сущностей и атрибутами отсутствует. Атрибут является таковым только в связи с типом сущности. В другом контексте атрибут может выступать как самостоятельная сущность. Например, для автомобильного завода цвет — это только атрибут продукта производства, а для лакокрасочной фабрики цвет — тип сущности.

Ключом называют минимальный набор атрибутов, по значениям которых можно однозначно найти требуемый экземпляр сущности. Минимальность означает, что исключение из набора любого атрибута не позволяет идентифицировать сущность по оставшимся. Для сущности РАСПИСАНИЕ ключом является атрибут Номер_рейса или набор: Пункт_отправления, Время_вылета и Пункт_назначения (при условии, что из пункта в пункт вылетает в каждый момент времени один самолёт).

Связь — это ассоциирование двух или более сущностей. Если бы назначением базы данных было только хранение отдельных, не связанных между собой данных, то ее структура могла бы быть очень простой. Однако одно из основных требований к организации базы данных — это обеспечение возможности отыскания одних сущностей по значениям других, для чего необходимо установить

между ними определенные связи. А так как в реальных базах данных нередко содержатся сотни или даже тысячи сущностей, то теоретически между ними может быть установлено более миллиона связей. Наличие такого множества связей и определяет сложность инфологических моделей.

При построении инфологических моделей можно использовать язык *ER-диаграмм* (от англ. *Entity—Relationship*, т.е. сущность—связь). В них сущности изображаются помеченными прямоугольниками, ассоциации — помеченными ромбами или шестиугольниками, атрибуты — помеченными овалами, а связи между ними — ненаправленными ребрами, над которыми может проставляться степень связи (1 или буква, заменяющая слово «много») и необходимое пояснение.

Между двумя сущностям, например, A и B возможны четыре вида связей:

1. *связь «один-к-одному» (1:1)*: одному экземпляру сущности *A* соответствует 1 или 0 представителей сущности *B* (рисунок 2.2):

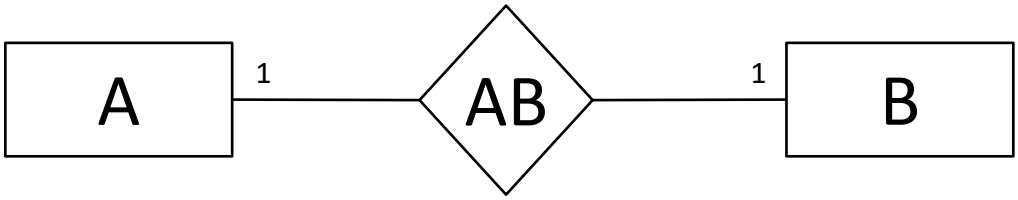

Рисунок 2.2. Связь «один-к-одному»

2. *связь «один-ко-многим» (1:N)*: одному экземпляру сущности *A* соответствует 0, 1 или несколько представителей сущности *B* (рисунок 2.3):

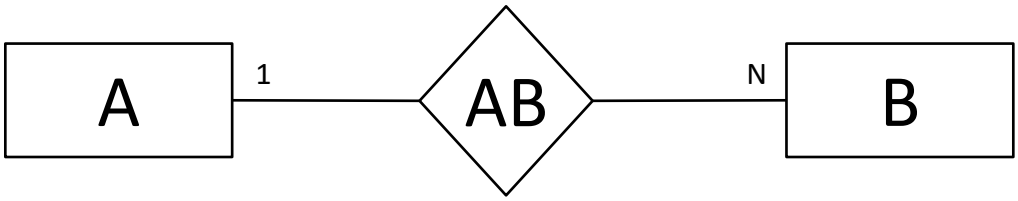

Рисунок 2.3. Связь «один-ко-многим»

3. *связь «многие-к-одному» (N:1)*: аналогична связи «один-ко-многим», если сущности поменять местами, поэтому её обычно не рассматривают как отдельный вид связи;
4. *связь «многие-ко-многим» (N:M)*: является комбинацией связей «один-ко-многим» и «многие-к-одному» (рисунок 2.4):

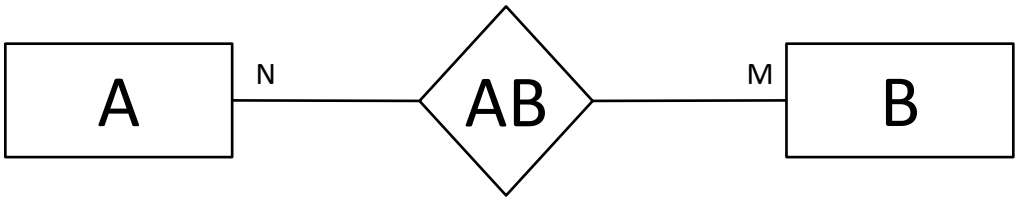

Рисунок 2.4. Связь «многие-ко-многим»

Существуют также более сложные связи, которые встречаются редко и зачастую сводятся к комбинации основных видов связей.

Обычно ER-диаграммы, или диаграммы «сущность—связь», используют только для описания связей между сущностями, поскольку добавление атрибутов существенно увеличивает размер диа-

граммы, что приводит к уменьшению её наглядности и удобочитаемости. Атрибуты каждой сущности обычно указывают в виде отдельного описания, которое можно выполнить как приложение к диаграмме.

Рассмотрим пример модели данных, описывающей библиотеку. Для начала необходимо определить сущности, информацию о которых необходимо хранить в базы данных. Пусть нас интересуют следующие сущности:

- книга — содержит информацию об отдельном экземпляре книги;
- автор — содержит информацию об авторе, книги которого есть в библиотеке;
- предметная область — содержит информацию о предметной области, которой посвящены книги;
- зал — содержит информацию о зале библиотеки, в котором находятся только те книги, которые посвящены одной из предметных областей;
- читатель — содержит информацию о читателе, который может брать книги из библиотеки.

Связи между этими сущностями можно построить, исходя из истинности следующих высказываний:

- Одна книга может быть написана одним или несколькими авторами.
- Один и тот же автор может написать одну или несколько книг.
- Каждая книга может быть посвящена только одной предметной области.
- Каждая предметная область находится только в одном зале, который посвящён только этой предметной области.
- В любой момент времени каждую книгу может взять только один читатель, либо она находится в библиотеке.
- Каждый читатель может быть записан в одном или нескольких залах и может одновременно брать одну или несколько книг.

Теперь на основе имеющейся информации можно составить ER-диаграмму (рисунок 2.5), которую можно будет использовать для проектирования структуры базы данных.

Как видно из диаграммы, сущность КНИГА связана с сущностью ЗАЛ через сущность ПРЕДМЕТНАЯ ОБЛАСТЬ, поскольку между сущностями есть связь «один-к-одному». В этом случае связь КНИГА—ЗАЛ не нужна, так как она будет избыточной. При этом, может показаться избыточной связь ЧИТАТЕЛЬ—ЗАЛ, но она необходима, так как читатель должен записаться в зал, чтобы брать в нём книги.

Остаётся определить атрибуты, которые характеризуют экземпляры каждой сущности. Предположим, что из всего множества сведений, которые можно получить о каждой сущности, в нашей базе данных нужно будет хранить только следующую информацию:

- КНИГА — Внутренний код экземпляра, Название, Год издания, Количество страниц, ISBN;
- АВТОР — Фамилия, Имя, Отчество, Год рождения;

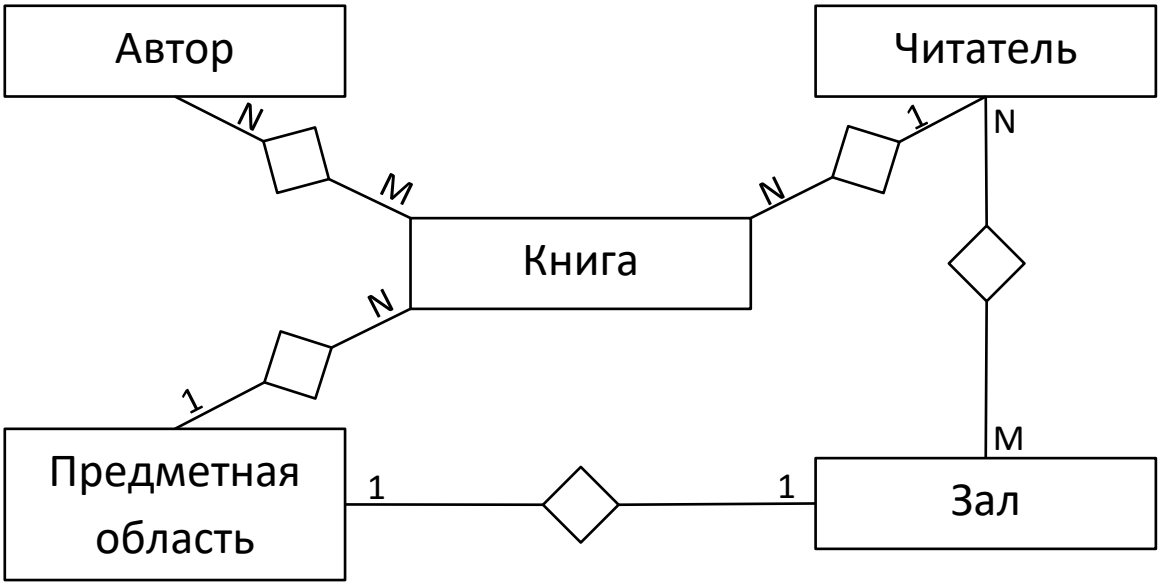

Рисунок 2.5. ER-диаграмма описания библиотеки

- ПРЕДМЕТНАЯ ОБЛАСТЬ — Название, Описание;
- ЗАЛ — Название, Расположение;
- ЧИТАТЕЛЬ — Фамилия, Имя, Отчество, Год рождения, Адрес.

Таким образом, мы получили инфологическую модель предметной области «Библиотека», на основе которой можно спроектировать структуру базы данных.

Краткие итоги

1. Создание базы данных начинается с анализа предметной области и выявления требований к ней отдельных пользователей. Вопросами проектирования базы данных обычно занимается администратор базы данных.
2. Архитектура СУБД включает три уровня: инфологический, даталогический и физический. Анализ предметной области предполагает построение инфологической модели данных.
3. Самой распространённой инфологической моделью данных является модель «сущность—связь». Она изображается в виде ER-диаграммы.
4. Диаграмма «сущность—связь» содержит сущности, связи между ними и атрибуты сущностей.
5. Сущность — это любой различимый объект, информацию о котором необходимо хранить в базе данных. Атрибуты являются характеристиками сущностей.
6. Существует три основных вида связей: «один-к-одному», «один-ко-многим» и «многие-ко-многим». Другие виды связей сводятся к комбинации основных связей.

Глава 3. Реляционная модель данных

Основные понятия

Реляционная модель важна по двум причинам. Во-первых, поскольку конструкции реляционной модели имеют широкий и общий характер, она позволяет описывать структуры баз данных независимым от СУБД образом. Во-вторых, реляционная модель является основой практически всех СУБД.

Термин «реляционная» происходит от англ. слова *relation* — отношение. Понятие «отношение» является центральным в реляционной модели.

Отношение (*relation*) — это двумерная таблица. Каждая строка в таблице содержит данные, относящиеся к некоторой сущности или какой-то её части. Каждый столбец таблицы описывает какой-либо атрибут этой сущности. Иногда строки называются *кортежами* (*tuples*), а столбцы — *атрибутами* (*attributes*).

Термины *отношение*, *кортеж* и *атрибут* пришли из реляционной алгебры, которая является теоретическим источником этой модели. Однако вместо этих терминов часто используются термины *таблица* (*table*), *строка* (*row*) и *столбец* (*column*).

Чтобы таблица была отношением, она должна удовлетворять определённым ограничениям:

- значения в ячейках таблицы должны быть одиночными — ни повторяющиеся группы, ни массивы не допускаются;
- все записи в столбце должны быть одного типа. Например, если третий столбец первой строки таблицы содержит номер сотрудника, то и во всех остальных строках таблицы третий столбец также должен содержать номер сотрудника;
- каждый столбец имеет уникальное имя. Порядок столбцов в таблице несуществен;
- в отношении не должно быть двух одинаковых строк, и порядок строк не имеет значения.

Каждое отношение имеет имя, уникальное в пределах базы данных. Имя отношения и список его атрибутов составляют *структуру отношения*.

Функциональная зависимость (*functional dependency*) — это связь между атрибутами. Предположим, что если мы знаем значение одного атрибута, то можем вычислить (или найти) значение другого атрибута.

Например, если нам известен номер счёта клиента, то мы можем определить состояние его счёта. В таком случае мы можем сказать, что атрибут СостояниеСчётаКлиента функционально зависит от атрибута НомерСчётаКлиента.

Говоря более общим языком, атрибут Y функционально зависит от атрибута X, если значение X определяет значение Y. Другими словами, если нам известно значение X, мы можем определить значение Y.

Уравнения выражают функциональные зависимости. Например, если мы знаем цену и количество приобретённого товара, мы можем определить стоимость покупки по следующей формуле:

Стоимость = Цена × Количество

В этом случае мы могли бы сказать, что атрибут Стоимость функционально зависит от атрибутов Цена и Количество.

Функциональные зависимости между атрибутами в отношении обычно не выражаются уравнениями. Пусть, например, каждому студенту присвоен уникальный идентификационный номер, и у каждого студента есть одна и только одна специальность. Имея номер студента, мы можем узнать его специальность, поэтому атрибут Специальность функционально зависит от атрибута НомерСтудента. Или рассмотрим компьютеры в вычислительной лаборатории. Каждый компьютер имеет конкретный размер основной памяти, поэтому атрибут ОбъёмПамяти функционально зависит от атрибута СерийныйНомерКомпьютера.

В отличие от случая с уравнением, такие функциональные зависимости нельзя разрешить при помощи арифметики; вместо этого они хранятся в базе данных. Фактически, можно утверждать, что базу данных стоит иметь только ради хранения и выдачи функциональных зависимостей.

Функциональные зависимости обозначаются следующим образом:

НомерСтудента > Специальность

СерийныйНомерКомпьютера > ОбъёмПамяти

Первое выражение читается так: «атрибут НомерСтудента функционально определяет атрибут Специальность», «атрибут НомерСтудента определяет атрибут Специальность» или «атрибут Специальность зависит от атрибута НомерСтудента». Атрибуты по правую сторону от стрелки называются *детерминантами* (*determinants*).

Как уже говорилось, если номер студента определяет специальность, то каждому номеру студента соответствует только одна специальность. Между тем, одной и той же специальности может соответствовать более одного номера студента. Пусть студент под номером 123 специализируется на бухгалтерском учёте. Тогда для любого отношения, где присутствуют столбцы НомерСтудента и Специальность, выполнится условие: если НомерСтудента = 123, то Специальность = БухгалтерскийУчёт. Обратное, однако, неверно: если Специальность = БухгалтерскийУчет, то атрибут НомерСтудента может принимать различные значения, так как на бухгалтерском учёте может специализироваться много студентов. Следовательно, мы можем сказать, что связь атрибутов НомерСтудента и Специальность имеет вид N:l. В общем случае можно утверждать, что если A определяет B, связь между значениями A и B имеет вид N:l.

В функциональные зависимости могут быть вовлечены группы атрибутов. Рассмотрим отношение ОЦЕНКИ (НомерСтудента, Дисциплина, Оценка). Комбинация номера студента и дисциплины определяет оценку. Такая функциональная зависимость записывается следующим образом:

(НомерСтудента, Дисциплина) > Оценка

Заметьте, что для определения оценки требуется как номер студента, так и дисциплина. Мы не можем разделить эту функциональную зависимость, поскольку ни номер студента, ни дисциплина не определяют оценку сами по себе.

Обратите внимание на следующее различие. Если X > (Y, Z), то X > Y и X > Z. Например, если НомерСтудента > (ИмяСтудента, Специальность), то НомерСтудента > ИмяСтудента и НомерСтудента > Специальность. Но если (X, Y) > Z, то в общем случае неверно, что X > Y или X > Z. Следовательно,

если (НомерСтудента, Дисциплина) > Оценка, то ни номер студента, ни дисциплина как таковые оценку не определяют.

Ключ (*key*) — это группа из одного или более атрибутов, которая уникальным образом идентифицирует строку. Рассмотрим отношение СЕКЦИЯ, имеющее атрибуты НомерСтудента, Секция и Плата. Строка этого отношения содержит информацию о том, что студент посещает определённую секцию за определённую плату. Предположим, что студент одновременно не может посещать более одной секции. В этом случае значение атрибута НомерСтудента идентифицирует единственную строку, поэтому данный атрибут является ключом.

Ключи могут также составляться из нескольких атрибутов. Например, если студентам разрешено одновременно посещать более одной секции, то один и тот же номер студента может появиться в разных строках таблицы, поэтому атрибут НомерСтудента не будет уникальным образом определять строку. Для этого потребуется какое-то сочетание атрибутов; возможно, это будет комбинация (НомерСтудента, Секция).

Допустим, в ходе опроса пользователей мы выяснили, что студентам разрешено одновременно быть записанными в несколько спортивных секций. Как мы отметили, НомерСтудента не является ключом этого отношения. Например, студент с номером 100 посещает секцию гольфа и секцию лыж, и атрибут НомерСтудента со значением 100 появляется в двух различных строках. Фактически, для этого отношения ни один атрибут сам по себе не является ключом, то есть ключ должен состоять из двух или более атрибутов.

Рассмотрим, какие комбинации из двух атрибутов возможны для данной таблицы. Таких комбинаций имеется три: (НомерСтудента, Секция), (НомерСтудента, Плата) и (Секция, Плата). Чтобы быть ключом, группа атрибутов должна уникальным образом идентифицировать строку. Допустим, плата за абонемент в различных секциях может совпадать. Поскольку это так, комбинация (НомерСтудента, Плата) не может уникальным образом определить строку. Например, студент с номером 100 может посещать две различных секции, каждая из которых стоит $200. Это означает, что комбинация (100, $200) возникла бы в таблице дважды, поэтому такая комбинация не может быть ключом.

Аналогичным образом можно определить, что сочетание (Секция, Плата) также не может являются ключом. А вот сочетание (НомерСтудента, Секция) может уникальным образом определять строку таблицы, но только если не требуется вести записи об истории посещения секций студентом. То есть, если в базе данных необходимо хранить сведения только о посещаемых в настоящий момент секциях, то комбинация (НомерСтудента, Секция) может уникальным образом определить строку отношения и, следовательно, эта комбинация является ключом данного отношения. Если необходимо также хранить записи о секциях, которые посещались в прошлом, то отношение имело бы дублирующиеся строки. Поскольку это недопустимо по определению отношения, нам пришлось бы добавить другие атрибуты, например атрибут Дата.

Из этого примера можно заключить, что каждое отношение имеет *минимум один ключ*. Это утверждение должно быть верным, поскольку ни одно отношение не может иметь одинаковых строк, и, следовательно, в крайнем случае, ключ будет состоять из всех атрибутов отношения.

Аномалии модификации

Не все возможные отношения одинаково желательны. Таблица, отвечающая минимальному определению отношения, может иметь неэффективную или неподходящую структуру. Для некоторых

отношений изменение данных может привести к нежелательным последствиям, называемым *аномалиями модификации* (*modification anomalies*). Аномалии могут быть устранены путём разбиения исходного отношения на два или более новых отношения. В большинстве случаев переопределённые, или нормализованные, отношения являются более предпочтительными.

Суть аномалий модификации заключается в том, что некоторые факты, которые относятся к одной сущности, зависят от другой сущности.

Например, если в отношении СЕКЦИЯ удалить строку, где записаны данные о студенте с номером 100, который один посещает секцию лыж, мы потеряем не только информацию о том, что этот студент является лыжником, но и тот факт, что абонемент в секцию лыж стоит $200. Это называется *аномалией удаления* (*deletion anomaly*) — то есть, удаляя факты, относящиеся к одной сущности (студент с номером 100 является лыжником), мы непроизвольно удаляем факты, относящиеся к другой сущности (плата за абонемент в секцию лыж составляет $200). Выполнив одну операцию удаления, мы теряем информацию о двух сущностях.

На примере того же самого отношения можно продемонстрировать *аномалию вставки* (*insertion anomaly*). Предположим, мы хотим записать в базу данных тот факт, что абонемент в секцию подводного плавания стоит $175, однако мы не можем ввести эти данные в отношение СЕКЦИЯ, пока хотя бы один студент не запишется в секцию подводного плавания. Получаем аномалию вставки — то есть, мы не можем записать в таблицу некоторый факт об одной сущности, не указав дополнительно некоторый факт о другой сущности.

Таким образом, описанное отношение СЕКЦИЯ имеет явные проблемы. Аномалию удаления и аномалию вставки можно устранить, разделив отношение СЕКЦИЯ на два отношения, каждое из которых будет содержать информацию на определённую тему. Например, в одно отношение мы поместим атрибуты НомерСтудента и Секция (назовем это отношение СТУДЕНТ-СЕКЦИЯ), а в другое — атрибуты Секция и Плата (назовем это отношение СЕКЦИЯ-ПЛАТА).

Теперь, если мы удалим запись о студенте с номером 100 из таблицы СТУДЕНТ-СЕКЦИЯ, мы уже не потеряем тот факт, что абонемент в секцию лыж стоит $200. Более того, мы можем добавить в таблицу СЕКЦИЯ-ПЛАТА секцию подводного плавания с указанием стоимости абонемента, не дожидаясь, пока кто-либо запишется в эту секцию. Таким образом, аномалии удаления и вставки устранены.

Однако, разбиение одного отношения на два имеет один недостаток. Предположим, что студент хочет записаться в несуществующую секцию. Например, студент с номером 250 хочет записаться в секцию ракетбола. Мы можем вставить соответствующую строку в отношение СТУДЕНТ-СЕКЦИЯ (строка будет содержать запись (250, Ракетбол)), но эта секция отсутствует в отношении СЕКЦИЯ-ПЛАТА. Таким образом, возникает неопределённость: нельзя определить информацию о сущности, которая соответствует другой сущности.

Допустим, секции могут существовать и до того, как кто-либо в них запишется, однако студент не может записаться в секцию, для которой не указан размер платы за абонемент (то есть в секцию, данные о которой отсутствуют в таблице СЕКЦИЯ-ПЛАТА). При проектировании базы данных можно задокументировать это ограничение любым из следующих способов: «множество значений атрибута Секция в таблице СТУДЕНТ-СЕКЦИЯ является подмножеством множества значений атрибута Секция в таблице СЕКЦИЯ-ПЛАТА», «СТУДЕНТ-СЕКЦИЯ [Секция] является подмножеством СЕКЦИЯ-ПЛАТА [Секция]» либо «СТУДЕНТ-СЕКЦИЯ [Секция] с СЕКЦИЯ-ПЛАТА [Секция]».

Квадратные скобки в этой записи обозначают столбец данных, извлекаемый из отношения. Смысл этих выражений в том, что атрибут Секция в отношении СТУДЕНТ-СЕКЦИЯ может принимать только те значения, которые имеет атрибут Секция в отношении СЕКЦИЯ-ПЛАТА. Это означает также, что прежде чем ввести название какой-то секции в отношение СТУДЕНТ-СЕКЦИЯ, мы должны проверить, что такое название уже существует в отношении СЕКЦИЯ-ПЛАТА. Ограничения, подобные этому, называются *ограничениями ссылочной целостности* (*referential integrity constraints*), или *ограничениями целостности по внешнему ключу* (*interrelation constraints*).

Аномалии, присутствующие в отношении СЕКЦИЯ, можно интуитивно описать следующим образом: проблемы возникают из-за того, что отношение СЕКЦИЯ содержит факты, относящиеся к двум различным темам.

1. Кто из студентов какую секцию посещает.
2. Какова плата за абонемент в каждой из секций.

Когда мы добавляем новую строку, нам приходится добавлять информацию, затрагивающую две различные темы; точно так же, когда мы удаляем строку, мы вынуждены удалять данные, относящиеся сразу к двум темам.

Каждое отношение должно иметь одну-единственную тему. Любое отношение, содержащее две или более темы, следует разбить на два или более отношения, каждое из которых будет содержать одну тему. Этот процесс составляет суть *нормализации*. Когда мы обнаруживаем отношение с аномалиями модификации, мы устраняем эти аномалии, разбивая отношение на два или более новых отношения, каждое из которых содержит факты, относящиеся к одной теме.

Однако не стоит забывать, что всякий раз, когда мы разбиваем отношение, мы, возможно, порождаем ограничение ссылочной целостности. Поэтому следует обязательно проверять наличие таких ограничений каждый раз при разбиении отношения на два или более новых.

Нормальные формы

Отношения можно классифицировать по типам аномалий модификации, которым они подвержены. В 1970-х годах теоретики реляционных баз данных постепенно сокращали количество этих типов. Кто-то находил аномалию, классифицировал её и думал, как предотвратить её возникновение. Каждый раз, когда это происходило, критерии построения отношений совершенствовались. Эти классы отношений и способы предотвращения аномалий называются *нормальными формами* (*normal forms*). В зависимости от своей структуры, отношение может быть в первой, во второй или в какой-либо другой нормальной форме.

В своей работе, последовавшей за эпохальной статьей 1970 г., Кодд и другие определили первую, вторую и третью нормальные формы (1НФ, 2НФ и 3НФ). Позднее была введена нормальная форма Бойса-Кодда (НФБК), а затем были определены четвёртая и пятая нормальные формы. Эти нормальные формы являются вложенными, то есть отношение во второй нормальной форме является также отношением в первой нормальной форме, а отношение в 5НФ (пятая нормальная форма) находится одновременно в 4НФ, НФБК, 3НФ, 2НФ и 1НФ.

Первая нормальная форма (1НФ). Отношение находится в 1НФ тогда и только тогда, когда в любом допустимом значении этого отношения каждый её кортеж содержит только одно значение для каждого из атрибутов.

О любой таблице данных, удовлетворяющей определению отношения, говорят, что она находится в первой нормальной форме. Для того, чтобы таблица была отношением, должно выполняться следующее: ячейки таблицы должны содержать одиночные значения и в качестве значений не допускаются ни повторяющиеся группы, ни массивы. Все записи в одном столбце (атрибуте) должны иметь один и тот же тип. Каждый столбец должен иметь уникальное имя, но порядок следования столбцов в таблице несуществен. Наконец, в таблице не может быть двух одинаковых строк, и порядок следования строк несуществен.

Как было показано ранее, отношения в первой нормальной форме могут иметь аномалии модификации. Чтобы устранить эти аномалии, можно разбить отношение на два или более новых отношения. При этом, новые отношения оказываются в некоторой другой нормальной форме, а в какой именно, зависит от того, какие аномалии мы устранили, а также от того, каким аномалиям подвержены получившиеся отношения.

Вторая нормальная форма (2НФ). Отношение находится в 2НФ тогда и только тогда, когда оно находится в 1НФ и каждый неключевой атрибут неприводимо зависит от его первичного ключа.

В соответствии с этим определением, если отношение имеет в качестве ключа одиночный атрибут, то оно автоматически находится во второй нормальной форме. Поскольку ключ является одиночным атрибутом, то по умолчанию каждый неключевой атрибут зависит от всего ключа, и частичных зависимостей быть не может. Таким образом, вторая нормальная форма представляет интерес только для тех отношений, которые имеют композитные ключи.

Отношение СЕКЦИЯ может быть разбито на два отношения во второй нормальной форме — это рассмотренные ранее отношения СТУДЕНТ-СЕКЦИЯ и СЕКЦИЯ-ПЛАТА. Мы знаем, что новые отношения находятся во второй нормальной форме, поскольку оба они имеют в качестве ключей одиночные атрибуты.

Третья нормальная форма (3НФ). Отношение находится в 3НФ тогда и только тогда, когда оно находится во 2НФ и каждый неключевой атрибут нетранзитивно зависит от его первичного ключа.

Определение 3НФ исключает *транзитивную зависимость*, которая может наблюдаться во 2НФ. Например, в отношении СЕКЦИЯ атрибут НомерСтудента определяет атрибут Секция, а Секция определяет атрибут Плата, то есть между атрибутами НомерСтудента и Плата существует транзитивная зависимость. Разбиение отношения СЕКЦИЯ на отношения СТУДЕНТ-СЕКЦИЯ и СЕКЦИЯ-ПЛАТА устраняет транзитивную зависимость.

Каждый кортеж отношения в 3НФ состоит из значения первичного ключа, идентифицирующего некоторую сущность, и набора из нуля и более взаимно независимых атрибутов, некоторым образом описывающих эту сущность.

3НФ часто рассматривается как основная, к которой стремятся преобразовать отношения, чтобы построить структуру базы данных, избежав при этом появление основных видов аномалий. Однако, при этом могут возникать другие аномалии, но зачастую они не так критичны, как описанные ранее, поэтому остальные нормальные формы, которые направлены на устранение этих аномалий, мы рассматривать не будем.

Проектирование структуры базы данных

Процесс создания структуры базы данных можно свести к процедуре *нормализации данных*, суть которого заключается в следующем. В предметной области выделяются сущности и их атрибуты, и эти сущности последовательно преобразуются так, чтобы они соответствовали 1НФ, затем — 2НФ и т.д. В результате получаются связанные таблицы, которые можно реализовать, используя выбранную СУБД.

Однако, если на этапе анализа предметной области была создана модель данных, например, модель «сущность—связь», то её можно преобразовать в табличную структуру, которая будет удовлетворять требованиям 3НФ.

Для начала рассмотрим несколько важных определений, которые играют важную роль в базах данных. Для удобства далее будем использовать термины «таблица», «столбец» и «строка» вместо терминов «отношение», «атрибут» и «кортеж».

Связь между двумя таблицами осуществляется по их ключевым полям. При этом, одна из таблиц называется *главной* (*master*), а другая — *подчинённой* (*detail*). Связь описывает логическую зависимость между записями таблиц, управление которой осуществляет СУБД.

Первичный ключ (*primary key*) — это набор из одного или нескольких столбцов таблицы, значения которых однозначно идентифицируют каждую строку этой таблицы. Обычно у каждой таблицы должен быть первичный ключ, хотя в некоторых случаях его не задают.

Вторичный, или ***внешний ключ*** (*foreign key*) — это набор из одного или нескольких столбцов одной таблицы (подчинённой), которая имеет связь с другой таблицей (главной). При этом, количество столбцов, которые составляют внешний ключ подчинённой таблицы, должно совпадать с количеством столбцов, образующих первичный ключ главной таблицы.

Строку таблицы часто называют ***записью***. Можно сказать, что каждая таблица базы данных может содержать 0 или более записей (или строк), а каждая запись содержит ***поля***, которые соответствуют столбцам этой таблицы. Соответственно, ***значение поля*** — это значение соответствующего столбца одной из строк таблицы.

Если между таблицами установлена связь, то для каждой записи подчинённой таблицы значения полей, соответствующих внешнему ключу, должны совпадать со значениями полей первичного ключа связанной записи главной таблицы. Однако, для любой записи таблицы некоторые её поля могут не содержать значений, если это разрешено в соответствующих столбцах таблицы. В этом случае говорят, что поле содержит *пустое значение*. Если все поля внешнего ключа одной из записей подчинённой таблицы содержат пустое значение, то такая запись не связана ни с какой записью главной таблицы.

В прошлой главе был описан пример создания ER-диаграммы для описания библиотеки (см. рисунок 2.5). Рассмотрим процесс её преобразования в связанные таблицы.

Каждая сущность представляется в виде таблицы, столбцы которой соответствуют атрибутам этой сущности. Если таблица будет главной таблицей в какой-либо связи, то для неё обязателен первичный ключ, в противном случае его можно не задавать.

Под первичный ключ таблицы необходимо выбрать один или несколько столбцов, однозначно идентифицирующих все записи этой таблицы. Рекомендуется, чтобы он содержал как можно меньше столбцов и занимал как можно меньше памяти, так как значения полей первичного ключа будут дублироваться в одной или нескольких записях подчинённой таблицы, и для экономии памяти их рекомендуется свести к минимуму.

В таблице КНИГА в качестве первичного ключа можно взять столбец *Внутренний код экземпляра*, так как предполагается, что это будет либо целое число, либо короткая строка фиксированного размера, и это значение для каждой книги будет различным.

Для таблицы АВТОР можно было бы определить первичный ключ из столбцов *Фамилия*, *Имя* и *Отчество*, но это решение имеет два существенных недостатка. Во-первых, нельзя гарантировать, что в таблице не будет двух человек, у которых совпадают и фамилия, и имя, и отчество, но отличаются другие параметры. Во-вторых, как было сказано выше, рекомендуется выбирать минимальное количество столбцов, поскольку их значения будут дублироваться, и в этом случае база данных будет занимать гораздо больше памяти, чем хотелось бы.

Наилучшим решением в этом случае будет добавление к этой таблице дополнительного столбца *Идентификатор*, который не соответствует ни одному атрибуту сущности АВТОР. Цель этого столбца — однозначная идентификация каждой записи таблицы. Значения полей, соответствующих этому столбцу, для каждой записи может генерироваться автоматически — такая возможность есть во всех СУБД, поэтому такое поле идеально подходит для первичного ключа.

Аналогичный подход можно применить и для остальных таблиц. Обратите внимание, что для таблиц ПРЕДМЕТНАЯ ОБЛАСТЬ и ЗАЛ можно было бы объявить первичным ключом столбец *Название*, но значения полей, связанных с этим полем, может быть довольно длинной строкой, и введение дополнительного короткого поля в конечном итоге минимизирует расход памяти, связанный с дублированием значений поля в записях подчинённых таблиц.

Теперь рассмотрим, как преобразовать связи между сущностями в связи между таблицами. Добавление в базу данных связей зависит от их типа.

Для связей «один-ко-многим» преобразование можно осуществить следующим образом. Для сущности, экземплярам которой соответствует один экземпляр связанной сущности, в таблицу добавляется столбец (или столбцы), который будет участвовать в формировании связи между таблицами и должен соответствовать столбцам первичного ключа главной таблицы.

Связь «один-к-одному» выполняется так же, как и связь «один-ко-многим». Единственное отличие — необходимость выбора главной и подчинённой таблицы.

Реализация связи «многие=ко-многим» осложняется тем, что в СУБД нет прямой поддержки таких связей. Однако, её можно преобразовать в две связи «один-ко-многим», если добавить таблицу с первичным ключом из двух столбцов, каждый из которых будет внешним ключом в связях с таблицами, соответствующих исходных сущностям. Первичный ключ нужен для того, чтобы связь между двумя записями связанных таблиц не повторялась.

Кроме этого, при создании таблиц рекомендуется, чтобы имена таблиц и столбцов содержали только латинские буквы и не содержали пробелов. Это позволит избежать проблем, связанных с особенностями СУБД, о которых будет рассказано позже. В остальном, выбор нотации зависит от программиста. Можно использовать, например, CamelCase-нотацию, стиль_через_подчёркивание.

Для формирования структуры базы данных удобно использовать специализированный редактор, например, программный продукт Microsoft Visio. Схема базы данных, соответствующая созданной ранее ER-диаграмме описания библиотеки, приведена на рисунке 3.1.

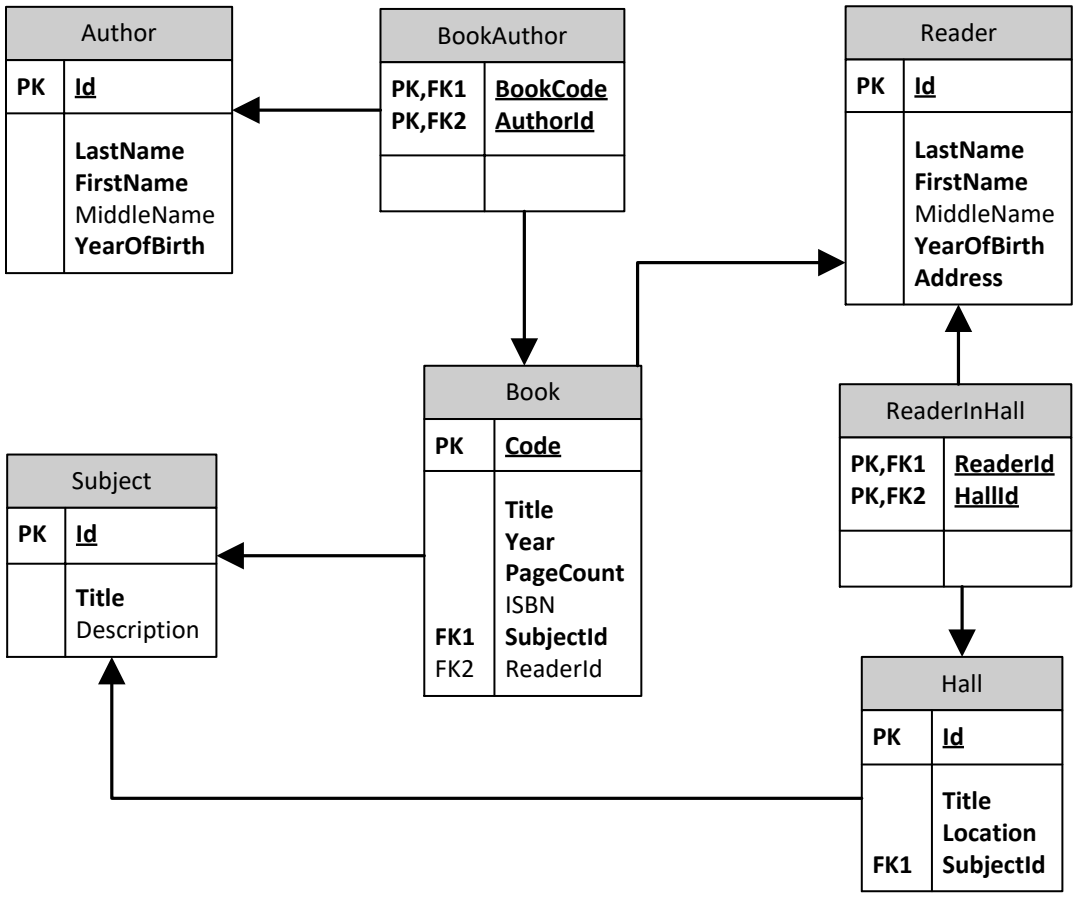

Рисунок 3.1. Структура базы данных «Библиотека»

В этой схеме стрелками изображены связи (стрелка ведёт от подчинённой таблицы к главной), жирным шрифтом — обязательные поля, PK обозначает первичный ключ, FKi — внешние ключи. Здесь таблицы BookAuthor и ReaderInHall служат для определения связей «многие-ко-многим».

Обратите внимание, что полученная база данных лишена аномалий и избыточности данных, все таблицы удовлетворяют требованиям 3НФ.

Однако, на практике иногда при разработке баз данных намеренно вводят избыточную информацию. Причины этого могут быть разные, и в следующих главах мы рассмотрим некоторые случаи, когда намеренное введение избыточности упрощает работу с данными. При этом, следует учитывать, что добавление избыточной информации требует решения дополнительных задач, связанных с поддержанием целостности и непротиворечивости хранимой в базе данных информации.

Краткие итоги

1. В реляционной модели центральным понятием является «отношение», обозначающее таблицу, в которой хранится информация о какой-либо сущности.
2. Каждый столбец таблицы описывает какой-либо атрибут сущности. Иногда строки называются *кортежами* (*tuples*), а столбцы — *атрибутами* (*attributes*).

3. Между атрибутами может существовать функциональная зависимость, которая позволяет по значению одного атрибута вычислить значение другого атрибута.
4. Группа из одного или более атрибутов, которая уникальным образом идентифицирует строку таблицы, называется *ключом*. Ключевые поля могут участвовать в формировании связи между таблицами.
5. В некоторых отношениях могут иметь место *аномалии модификации*, когда одна сущность хранится внутри другой сущности. Для устранения возможных аномалий отношения нужно приводить к одной из *нормальных форм*.
6. Существует несколько видов нормальных форм. Нормальные формы более высоких порядков включают нормальные формы низших порядков. Самой распространённой является третья нормальная форма.
7. Поля первичного ключа таблицы однозначно определяют записи в этой таблице.
8. Внешний ключ одной таблицы образует связь с первичным ключом другой таблицы. Последняя таблица является главной, другая — подчинённой.
9. Проектирование структуры базы данных можно осуществлять либо переводом отношений от одной нормальной формы к другой, либо на основе разработанной инфологической модели предметной области.
10. При преобразовании ER-диаграммы в табличную структуру сущности с их атрибутами преобразуются, соответственно, в таблицы и столбцы, связи «один-к-одному» и «один-ко-многим» — в связи между таблицами с добавлением в подчинённую таблицу столбцов внешнего ключа, а связь «многие-ко-многим» преобразуется в две связи «один-ко-многим» путём добавления связующей таблицы с двумя столбцами внешнего ключа, образующих первичный ключ.

Глава 4. Язык SQL. Создание структуры базы данных

SQL (*Structured Query Language* — структурированный язык запросов) — универсальный компьютерный язык, применяемый для создания, модификации и управления данными в реляционных базах данных. SQL основывается на исчислении кортежей.

В 1986 году Американский национальный институт стандартов (ANSI) представил свою первую версию стандарта, описанного в документе ANSI X3.135-1986 под названием «Database Language SQL» (Язык баз данных SQL). Неофициально этот стандарт SQL-86 получил название SQL1. Год спустя, была завершена работа над версией стандарта ISO 9075-1987 под тем же названием. Разработка этого стандарта велась под патронажем Технического Комитета ТС97 (англ. Technical Committee TC97), областью деятельности которого являлись процессы вычисления и обработки информации (англ. Computing and Information Processing). Именно его подразделение, именуемое как Подкомитет SC21 (англ. Subcommittee SC21) курировало разработку стандарта, что стало залогом идентичности стандартов ISO и ANSI для SQL1 (SQL-86).

Со временем к стандарту накопилось несколько замечаний и пожеланий, особенно с точки зрения обеспечения целостности и корректности данных, в результате чего в 1989 году данный стандарт был расширен, получив название SQL89. В частности, в него была добавлена концепция первичного и внешнего ключей. ISO-версия документа получила название ISO 9075:1989 «Database Language SQL with Integrity Enhancements» (Язык баз данных SQL с добавлением контроля целостности). Параллельно была закончена и ANSI-версия.

Сразу после завершения работы над стандартом SQL1 в 1987 году была начата работа над новой версией стандарта, который должен был заменить стандарт SQL89, получив название SQL2, поскольку дата принятия документа на тот момент была неизвестна. Таким образом, фактически SQL89 и SQL2 разрабатывались параллельно. Новая версия стандарта была принята в 1992 году, заменив стандарт SQL89. Новый стандарт, озаглавленный как SQL92, представлял собой по сути расширение стандарта SQL1, включив в себя множество дополнений имевшихся в предыдущих версиях инструкций.

Следующим стандартом стал SQL:1999 (SQL3). В настоящее время действует стандарт, принятый в 2003 году (SQL:2003) с небольшими модификациями, внесёнными позже.