1 Знакомство

Эта глава содержит сведения о построении базы данных и о подготовительных этапах работы: проектировании БД, установке и запуске MySQL. Следующий раздел даст вам общее представление об этой программе.

1.1. Что такое MySQL

MySQL — это свободно распространяемая СУБД, разработанная компанией MySQL AB (www.mysql.com). MySQL имеет клиент-серверную архитектуру: к серверу MySQL могут обращаться различные клиентские приложения, в том числе с удаленных компьютеров. Рассмотрим важнейшие особенности MySQL, благодаря которым эта программа приобрела популярность.

- MySQL — это СУБД с открытым кодом. Любой желающий может бесплатно скачать программу на сайте разработчика (http://dev.mysql.com/downloads/) и при необходимости доработать ее. Существует множество приложений MySQL, созданных и свободно распространяемых сторонними разработчиками. Однако для применения MySQL в коммерческом приложении необходимо приобрести коммерческую лицензированную версию программы у компании MySQL AB.
- MySQL — кроссплатформенная система. Ее можно использовать практически во всех современных операционных системах, в том числе Windows, Linux, Mac OS, Solaris, HP-UX и др. В этой книге мы рассмотрим работу с MySQL только в ОС Windows.
- MySQL имеет множество программных интерфейсов (API), благодаря которым к базе данных MySQL могут подключаться приложения, созданные с помощью C/C++, Eiffel, Java, Perl, PHP, Python, Tcl, ODBC, .NET и Visual Studio. В главе 4 вы узнаете, как обращаться к базе данных MySQL из PHP-, Perl- и Java-приложений.
- MySQL имеет отличные технические характеристики: многопоточность, многопользовательский доступ, быстродействие, масштабируемость (компания-раз-

работчик приводит пример MySQL-сервера, который работает с 60 тыс. таблиц, содержащими приблизительно 5 млрд строк).

- MySQL имеет развитую систему обеспечения безопасности и разграничения доступа на основе системы привилегий (гл. 5).

MySQL представляет собой реляционную СУБД, то есть систему управления реляционными базами данных. Поэтому для построения базы данных в MySQL нам потребуются базовые понятия теории реляционных баз данных. Этим понятиям посвящается следующий раздел.

1.2. Основные сведения о реляционных базах данных

Из этого раздела вы узнаете, как устроена реляционная база данных. Вначале мы рассмотрим таблицы, затем ключевые столбцы, связи между таблицами и, наконец, целостность данных в базе.

Таблицы

Реляционная база данных существует в виде таблиц, имеющих свои имена. На пересечении каждого столбца и каждой строки располагается одно значение.

Рассмотрим таблицу, содержащую сведения о клиентах компании (табл. 1.1).

Таблица 1.1. Customers (Клиенты)

id (идентификатор)	name (имя)	phone (телефон)	address (адрес)	rating (рейтинг)
533	ООО «Кускус»	313-48-48	ул. Смольная, д. 7	1000
534	Петров	7(929)112-14-15	ул. Рокотова, д. 8	1500
536	Крылов	444-78-90	Зеленый пр-т, д. 22	1000

Строки таблицы могут храниться в произвольной последовательности и не должны повторяться.

Каждый столбец таблицы имеет имя и *тип данных*, которому соответствуют все значения в столбце. Так, в нашем примере столбцы с именами `id` и `rating` — числовые, а с именами `name`, `phone` и `address` — символьные.

По существу, таблица реляционной базы данных представляет собой набор информации об однотипных объектах. При этом каждая строка содержит сведения об одном объекте, а каждый столбец — значения некоторого атрибута этих объектов. Например, строка с идентификационным номером 533 содержит информацию об объекте, у которого атрибут `name` (имя) имеет значение `ООО «Кускус»`, атрибут `phone` (телефон) — значение `313-48-48` и т. д.

Далее мы рассмотрим специальные столбцы таблицы — первичный и внешний ключи.

Первичный ключ

Строки таблицы неупорядочены и не имеют номеров, поэтому различить их можно только по содержащимся значениям. В связи с этим возникает необходимость рассмотреть понятие первичного ключа (primary key).

Первичный ключ — это минимальный набор столбцов, совокупность значений которых однозначно определяет строку. Это означает, что в таблице не должно быть строк, у которых значения во всех столбцах первичного ключа совпадают, при этом ни один столбец нельзя исключить из первичного ключа, иначе это условие нарушится.

На практике первичным ключом служит специальный столбец, значения которого автоматически задает СУБД. Например, в таблице `Customers` (Клиенты) (см. табл 1.1) это столбец `id` (идентификатор). Использовать такой искусственный первичный ключ значительно проще, чем естественный (основанный на атрибутах объекта). Например, в таблице `Customers` столбец `name` (имя) не может быть первичным ключом, так как имена клиентов могут совпадать; а первичный ключ из столбцов `name` (имя) и `phone` (телефон) был бы слишком громоздким. Дополнительными преимуществами искусственного ключа являются гарантированная уникальность значений (ее обеспечивает СУБД), постоянство значений (может меняться значение атрибута, но не значение искусственного ключа), а также числовой тип данных (поиск по числовым значениям выполняется намного быстрее, чем по символьным).

Еще одна функция первичного ключа — организация *связей* между таблицами.

Связи между таблицами. Внешний ключ

Реляционная база данных — это не просто набор таблиц. Объединить разрозненные фрагменты информации в единую структуру данных позволяют *связи* между таблицами, посредством которых строка одной таблицы сопоставляется строке (строкам) другой таблицы. Благодаря связям можно извлекать информацию одновременно из нескольких таблиц (например, выводить с помощью одного запроса и сведения о клиенте, и сведения о его заказах), избегать дублирования информации (не требуется в каждом заказе хранить адрес клиента), поддерживать полноту информации (не хранить сведения о заказанном товаре, если в базе данных отсутствует его описание) и многое другое.

Рассмотрим на примере, что такое связь между таблицами. Допустим, у нас есть таблицы *A* и *B*, и мы хотим их связать. Для этого в каждую строку таблицы *A* мы должны поместить некую информацию, позволяющую идентифицировать связанную с ней строку таблицы *B*. Эта информация называется *ссылкой*, а поля таблицы *A*, содержащие эту ссылку, — *внешними ключами*. Наверное, вы уже сами догадались, что в качестве ссылки используется первичный ключ таблицы *B*, поскольку именно его значения позволят однозначно идентифицировать нужную строку таблицы *B*. После того как мы во все строки таблицы *A* поместим ссылки на строки таблицы B, эти таблицы будут связаны. При этом таблица *A* будет называться *дочерней*, а таблица *B* — *родительской*.

Существует три типа связей, устанавливаемых между таблицами в базе данных.

- Связь «один ко многим».

  Этот тип связи используется чаще всего. В этом случае одна или несколько строк таблицы *A* ссылаются на одну из строк таблицы *B*.

  Для установки связи между таблицами в дочернюю таблицу добавляется *внешний ключ* (foreign key) — один или несколько столбцов, содержащих значения первичного ключа родительской таблицы (иными словами, во внешнем ключе хранятся *ссылки* на строки родительской таблицы).

  Рассмотрим таблицу, которая содержит сведения о заказах, сделанных клиентами, и является дочерней по отношению к таблице `Customers` (Клиенты) (табл. 1.2).

Таблица 1.2. Orders (Заказы)

id (идентификатор)	date (дата)	product_id (товар)	qty (количество)	amount (сумма)	customer_id (клиент)
1012	12.12.2007	5	8	4500	533
1013	12.12.2007	2	14	22 000	536
1014	21.01.2008	5	12	5750	533

  В таблице `Orders` внешним ключом является столбец `customer_id` (клиент), в котором содержатся номера клиентов из таблицы `Customers` (Клиенты). Таким образом, каждая строка таблицы `Orders` ссылается на одну из строк таблицы `Customers`. Например, строка с идентификационным номером 1012 содержит в столбце `customer_id` (клиент) значение 533: это означает, что заказ № 1012 сделан клиентом ООО «Кускус».

  Столбец `product_id` таблицы `Orders` также является внешним ключом — он содержит номера товаров из столбца `id` (идентификатор) таблицы `Products` (Товары). Таким образом, таблица `Orders` является дочерней по отношению к таблицам `Customers` и `Products`.

- Связь «один к одному».

  Такая связь между таблицами означает, что каждой строке одной таблицы соответствует одна строка другой таблицы, и наоборот. Например, если требуется хранить паспортные данные клиентов, можно создать таблицу `Passports` (Паспорта), связанную отношением «один к одному» с таблицей `Customers` (Клиенты).

  Таблицы, соединенные связью «один к одному», можно объединить в одну. Две таблицы вместо одной используют по соображениям конфиденциальности (например, можно ограничить доступ пользователей к таблице `Passports`), для удобства (если в единой таблице слишком много столбцов), для экономии дискового пространства (в дополнительную таблицу выносят те столбцы, которые часто бывают пустыми, тогда дополнительная таблица содержит значительно меньше строк, чем основная, и обе они занимают меньше места, чем единая таблица).

Связь «один к одному» может быть организована так же, как связь «один ко многим», — с помощью первичного ключа родительской таблицы и внешнего ключа дочерней. Другой вариант — связь посредством первичных ключей обеих таблиц, при этом связанные строки имеют одинаковое значение первичного ключа.

- Связь «многие ко многим».

  Этот тип связи в реляционной базе данных реализуется только с помощью вспомогательной таблицы. Например, если потребуется включить в заказ несколько наименований товаров, связь «многие ко многим» между таблицами `Orders` (Заказы) и `Products` (Товары) можно организовать с помощью вспомогательной таблицы `Items` (Позиции заказа), содержащей столбцы `product_id` (номер товара из таблицы `Products`), `qty` (количество товаров данного наименования в заказе) и `order_id` (номер заказа из таблицы `Orders`). При этом столбцы `product_id` и `qty` из таблицы `Orders` исключаются. Таким образом, таблица `Items` будет дочерней по отношению к таблицам `Orders` и `Products` и каждая строка таблицы `Items` будет соответствовать одному наименованию товара в заказе.

Как видим, реляционная база данных представляет собой весьма запутанную структуру, в которой все части (то есть записи) ссылаются на другие самым произвольным образом. А раз структура сложная, то неизбежны ее нарушения, происходящие по различным причинам, включая сбои программы, ошибки оператора и др. Последствия такого нарушения могут быть просто катастрофическими: скажем, что будет, если таблица заказов будет неверно ссылаться на таблицу товаров? Вся деятельность фирмы будет дезорганизована — вместо заказанного товара, допустим лопат, заказчику доставят топоры, а то и вовсе ничего, если ссылка на заказанный товар указывает на несуществующую строку таблицы товаров.

Итак, важнейшим понятием теории реляционных баз данных является *целостность данных*.

Целостность данных

Целостностью данных, хранимых в СУБД, называется их корректность и непротиворечивость.

Базовыми требованиями целостности, которые должны выполняться в любой реляционной базе данных, являются *целостность сущностей* и *целостность связей* (ссылочная целостность).

Целостность сущностей означает, что в каждой таблице есть первичный ключ — уникальный идентификатор строки. Первичный ключ не должен содержать повторяющихся и неопределенных значений. Например, если в таблицу `Customers` (Клиенты) добавить еще одну строку с идентификатором 533 (притом что одна строка с таким идентификатором уже существует в таблице), то целостность сущностей будет нарушена и невозможно будет определить, кому

из этих двух клиентов с одинаковыми идентификаторами принадлежат заказы №№ 1012 и 1014.

Целостность связей означает, что внешний ключ в дочерней таблице не содержит значения, отсутствующие в первичном ключе родительской таблицы. Иными словами, строка дочерней таблицы не должна ссылаться на несуществующую строку родительской таблицы. В отличие от первичного, внешний ключ может содержать неопределенные значения (NULL), и в этом случае целостность не нарушится. Например, в таблицу Orders (Заказы) добавлена строка, содержащая в столбце customer_id значение 999. Здесь нарушится целостность связи между таблицами Customers и Orders: с одной стороны, заказ не является «ничьим», так как в этом случае в столбце customer_id было бы установлено значение NULL, с другой стороны, невозможно выяснить имя и адрес клиента, сделавшего этот заказ.

Как видно из приведенных примеров, если целостность данных нарушена, то с ними невозможно нормально работать. Поэтому поддержание целостности данных является одной из основных функций любой СУБД.

Для поддержания целостности сущностей СУБД проверяет корректность значения первичного ключа при добавлении и изменении строк. Механизм поддержания ссылочной целостности более сложный. Помимо проверки корректности значения внешнего ключа при добавлении и изменении строк дочерней таблицы, необходимо также предотвратить нарушение ссылочной целостности при удалении и изменении строк родительской таблицы. Для этого существует несколько способов.

- Запрет (RESTRICT): если на строку родительской таблицы ссылается хотя бы одна строка дочерней таблицы, то удаление родительской строки и изменение значения первичного ключа в такой строке запрещаются. Например, не допускается удаление информации о клиенте из таблицы Customers (Клиенты), если у этого клиента есть зарегистрированные заказы, то есть строки в таблице Orders (Заказы), которые ссылаются на строку со сведениями об этом клиенте.
- Каскадное удаление/обновление (CASCADE): при удалении строки из родительской таблицы автоматически удаляются все ссылающиеся на нее строки дочерней таблицы; при изменении значения первичного ключа в строке родительской таблицы автоматически обновляется значение внешнего ключа в ссылающихся на нее строках дочерней таблицы.

  Например, при удалении записи о клиенте из таблицы Customers (Клиенты) автоматически удаляются сведения о заказах этого клиента, то есть соответствующие строки в таблице Orders (Заказы).
- Обнуление (SET NULL): при удалении строки и при изменении значения первичного ключа в строке значение внешнего ключа во всех строках, ссылающихся на данную, автоматически становится неопределенным (NULL).

  Например, при удалении записи о клиенте из таблицы Customers (Клиенты) заказы этого клиента автоматически становятся «ничьими», то есть

в соответствующих строках таблицы `Orders` (Заказы) в столбце `customer_id` (клиент) устанавливается значение `NULL`.

В СУБД MySQL способ поддержания целостности связи указывается при создании или изменении структуры дочерней таблицы.

С понятием целостности данных тесно связано понятие *транзакции*. Транзакцией называется группа связанных операций, которые должны быть либо все выполнены, либо все отменены. Если при выполнении одной из операций происходит ошибка или сбой, то транзакция отменяется. При этом все уже внесенные другими операциями изменения автоматически аннулируются и восстанавливается исходное состояние базы данных. Важнейшее применение транзакций — это объединение тех операций, которые, будучи выполнены по отдельности, могут нарушить целостность данных. Например, рассмотренная выше операция каскадного удаления выполняется как единая транзакция: строка родительской таблицы должна быть удалена вместе со всеми ссылающимися на нее строками дочерней таблицы, а если по каким-либо причинам одну из этих строк удалить невозможно, то не будет удалена ни одна из строк.

Теперь, когда вы ознакомились с основными понятиями теории реляционных баз данных, можно приступить к разработке собственной базы.

1.3. Проектирование базы данных

Построение базы данных (как и любой информационной системы, любого программного продукта) начинается с проектирования. В процессе его мы определяем задачи, для решения которых предназначена база данных, и создаем представление о данных и связях между ними.

Проектирование включает в себя следующие основные этапы.

- Определение требований к базе данных.

  В первую очередь, необходимо составить перечень требований, которым должна соответствовать проектируемая база данных. В этом разделе мы рассматриваем только функциональные требования. Другие требования (производительность, масштабируемость, надежность) также нужно учитывать, однако их выполнение во многом зависит от используемой СУБД.

  Например, при проектировании базы данных для торговой компании может выясниться, что отделу по работе с клиентами необходимо знать номера телефонов всех клиентов, отделу доставки нужен отчет, содержащий адрес клиента и список заказанных им товаров, отделу логистики — информация о том, какие товары в каком количестве были заказаны в прошлом месяце, и т. п. Эти требования и будут положены в основу проекта базы данных.

- Создание модели данных, соответствующей всем предъявленным требованиям.

  Для разработки модели данных на основе сформулированных требований можно использовать одну из двух противоположных стратегий.

- Проектирование «снизу вверх», от элемента к структуре: вначале определяется, какие именно атрибуты должны храниться в базе данных, затем группы атрибутов объединяются в объекты. Этот метод годится для небольших баз данных, в которых количество атрибутов невелико.
- Проектирование «сверху вниз» начинается с выделения высокоуровневых объектов и связей между ними, затем осуществляется декомпозиция объектов и последовательная детализация модели до уровня атрибутов. Для сложных баз данных с большим количеством атрибутов такой метод более эффективен, чем метод «снизу вверх».

В результате мы получим предварительную структуру базы данных: список объектов — таблиц и список атрибутов каждого объекта — столбцов таблицы. Например, на основе требований, приведенных в п. 1, можно построить модель данных, содержащую сведения о таких объектах, как клиенты, заказы и товары.

- Для клиентов: идентификатор, имя (или название организации), номер контактного телефона, адрес, а также рейтинг, используемый для расчета скидки.
- Для товаров: идентификатор, наименование, описание, название склада, где хранится этот вид товара, и адрес склада.
- Для заказов: дату заказа, идентификатор заказанного товара, количество товаров этого наименования, общую стоимость заказа с учетом скидки, идентификатор клиента, сделавшего заказ, и адрес клиента, куда нужно доставить заказ (здесь мы предполагаем, что каждый заказ может включать только одно наименование товара).

❑ Нормализация.

Нормализация базы данных заключается в минимизации избыточности данных. Нормализация позволяет уменьшить объем БД и устранить потенциальную противоречивость данных (например, если в базе данных одна и та же информация дублируется в нескольких местах, то при ее обновлении есть риск появления разночтений).

Результатом нормализации является приведение таблиц базы данных к одной из *нормальных форм*. На практике чаще всего используются три нормальные формы.

- Таблица находится в первой нормальной форме, если все атрибуты атомарны, то есть на пересечении любого столбца и строки находится значение, части которого не будут использоваться по отдельности.

  Ответ на вопрос, является ли атрибут атомарным, зависит от функциональных требований к базе данных. Рассмотрим, например, столбец `address` (адрес) из таблицы `Customers` (Клиенты) (см. табл. 1.1). Если адрес клиента будет использоваться только целиком, то этот столбец является атомарным. Если же потребуется получать из базы отдельно название города, улицы и т. п., то для приведения таблицы `Customers` к первой нормальной форме

столбец `address` следует разбить на столбцы `city` (город), `street` (улица), `building` (здание) и т. д.

- Таблица находится во второй нормальной форме, если она находится в первой нормальной форме и ни один из ее неключевых атрибутов не находится в функциональной зависимости от части первичного ключа.

  Это означает, что в таблице, в которой есть составной первичный ключ, значения остальных столбцов таблицы должны зависеть от значений *всех* столбцов первичного ключа. Если же есть столбцы, которые зависят только от *некоторых* столбцов первичного ключа, то для приведения таблицы во вторую нормальную форму необходимо перенести все эти столбцы в другую таблицу.

  Например, в нашей модели, построенной в п. 1, в таблице заказов первичным ключом может служить набор столбцов, содержащих дату заказа, идентификатор товара и идентификатор клиента (если мы допустим, что клиент не может сделать повторный заказ того же товара в тот же день, а может только изменить ранее сделанный заказ). Таким образом, для приведения таблицы заказов ко второй нормальной форме нужно исключить из таблицы адрес клиента, так как он зависит от идентификатора клиента, который является частью возможного первичного ключа. В противном случае адрес клиента будет повторяться в каждом заказе, что может привести к несогласованности данных. В частности, при изменении адреса клиента потребуется изменить адрес во всех заказах этого клиента. Если при выполнении такого массового обновления данных произойдет ошибка, то возможна ситуация, когда в некоторых заказах адрес будет изменен, а в некоторых останется прежним, и будет неясно, какой из адресов правильный. Нормализация таблицы позволяет избежать такой несогласованности.

ПРИМЕЧАНИЕ

> Атрибут A функционально зависит от группы атрибутов B, если значение атрибута A однозначно определяется набором значений группы атрибутов B, иными словами, в строках с одинаковым набором значений атрибутов группы B значение атрибута A также одинаково.

- Таблица находится в *третьей нормальной форме*, если она находится во второй нормальной форме и любой неключевой атрибут функционально зависит *только* от первичного ключа.

Например, в модели данных из п. 1 таблица, содержащая сведения о товарах, не находится в третьей нормальной форме, поскольку в ней имеется функциональная зависимость адреса склада от его названия. Таким образом, вам придется всякий раз при упоминании склада писать и его адрес, что приведет к многократному дублированию данных. Чтобы привести таблицу к третьей нормальной форме, все данные о складе нужно вынести в отдельную таблицу, которая будет родительской по отношению к таблице товаров.

Когда все таблицы базы данных приведены в третью нормальную форму, мы можем считать, что наша база данных нормализована, а информация о каждом факте хранится только в одном месте.

Итак, мы разработали логическую структуру базы данных, и можно переходить к созданию базы данных в СУБД MySQL. Если программа MySQL еще не установлена на вашем компьютере, из следующего раздела вы узнаете, как это сделать.

1.4. Установка и настройка MySQL

В этом разделе вы узнаете, как установить сервер MySQL и выполнить его начальную настройку. Начнем с нескольких советов по загрузке программы.

Загрузка MySQL

Как упоминалось ранее, дистрибутив MySQL можно бесплатно скачать с сайта компании-разработчика. Windows-версию MySQL вы найдете на веб-странице http://dev.mysql.com/downloads/mysql/5.0.html в одном следующих разделов:

- Windows downloads — здесь находятся ссылки на дистрибутивы MySQL для 32-разрядных операционных систем: Windows Vista/Server 2003/XP/Millennium Edition/2000/98/95;
- Windows x64 downloads — здесь располагаются ссылки на дистрибутивы MySQL для 64-разрядных операционных систем: Windows Vista x64/Server 2003 x64/XP x64.

В каждом из этих разделов представлены три варианта дистрибутива:

- Essentials — базовый вариант без опциональных компонентов. Включает мастер настройки (Configuration Wizard);
- ZIP/Setup.EXE — полный вариант, включающий опциональные утилиты, документацию и др., а также мастер настройки;
- Without installer — полный вариант, который не включает мастер настройки и требует ручной установки и настройки.

Рекомендуется использовать первый или второй вариант. Возможностей, предоставляемых первым вариантом, в большинстве случаев достаточно. В этой книге будет описана настройка MySQL только с помощью мастера настройки.

Если вы предпочитаете работать в графическом интерфейсе, а не в командной строке, то можете также скачать пакет графических утилит MySQL GUI Tools (http://dev.mysql.com/downloads/gui-tools/5.0.html), включающий три программы:

- MySQL Administrator — инструмент администрирования, конфигурирования, мониторинга, запуска/остановки сервера MySQL, управления пользователями и соединениями;
- MySQL Query Browser — инструмент создания, выполнения и оптимизации запросов в графической среде;

- MySQL Migration Toolkit — инструмент для переноса данных в MySQL из других реляционных баз данных (Oracle, Microsoft SQL Server, Microsoft Access, Sybase и др.).

ПРИМЕЧАНИЕ

В этой книге будет кратко рассказано, как пользоваться утилитами MySQL Administrator и MySQL Query Browser, а также описано выполнение операций с базой данных в командной строке.

Вы скачали необходимые дистрибутивы. Теперь приступим к установке СУБД MySQL 5.0.

Установка сервера MySQL

Чтобы установить на компьютере Windows-версию программы MySQL, выполните следующие действия.

1. Запустите мастер установки (Setup Wizard):
   - если вы скачали базовый вариант дистрибутива MySQL, то дважды щелкните на значке файла `mysql-essential-5.0.xxx-win32.msi`.;
   - если вы скачали полный вариант дистрибутива MySQL, то извлеките из архива файл `Setup.exe` и запустите его, дважды щелкнув на значке файла.
2. В начальном окне мастера установки (рис. 1.1) нажмите кнопку Next (Далее).

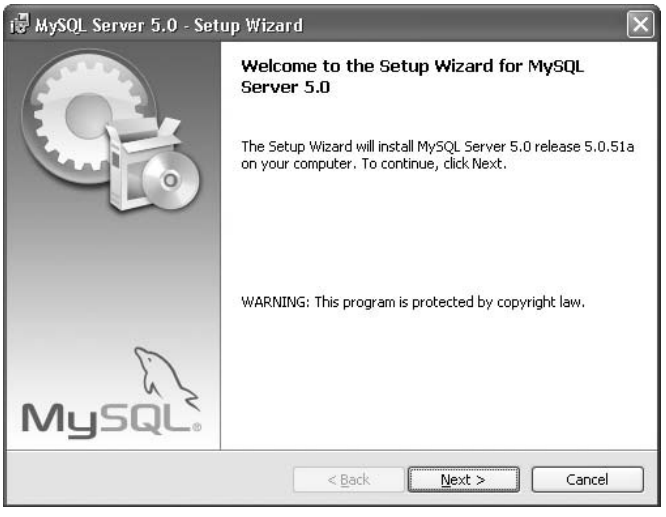

Рис. 1.1. Начальное окно мастера установки

3. Выберите тип установки (рис. 1.2):
   - Typical (Обычная) — будут установлены только основные компоненты MySQL: сервер и утилиты командной строки;

- Complete (Полная) — будут установлены все компоненты MySQL, в том числе библиотеки и заголовочные файлы;
- Custom (Выборочная) — вы сможете указать путь к каталогу, в котором будет установлена программа MySQL, и выбрать компоненты, которые требуется установить.

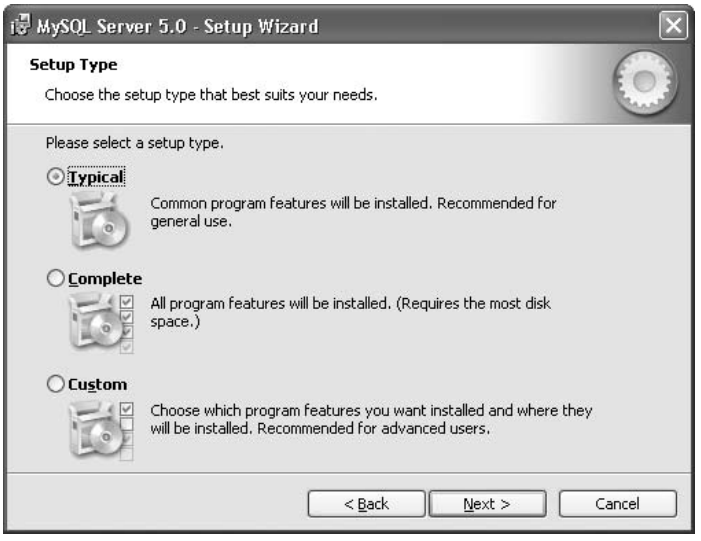

Рис. 1.2. Выбор типа установки

Рекомендуется выбрать вариант Custom (Выборочная), иначе выбор каталога установки будет недоступен.

Нажмите кнопку Next (Далее).

Если вы выбрали тип установки Typical (Обычная) или Complete (Полная), то следующий пункт пропустите.

Если вы выбрали тип установки Custom (Выборочная), то укажите параметры установки (рис. 1.3).

- Если необходимо включить в установку или исключить из нее какой-либо компонент, найдите его в дереве компонентов, щелкните на значке слева от названия компонента и в контекстном меню выберите нужное действие (— компонент будет установлен, — компонент не будет установлен). Если вы пока не знаете точно, какие компоненты вам потребуются, рекомендую оставить набор компонентов неизменным.
- Если необходимо изменить каталог установки, нажмите кнопку Change (Изменить). В появившемся окне (рис. 1.4) введите путь к каталогу в поле Folder name (Имя каталога) либо в поле Look in (Смотреть в) и выберите из списка нужный диск, а затем последовательно раскройте вложенные папки, пока не дойдете до нужной. Нажмите кнопку OK.