Цели работы:

1. Научиться проектировать базы данных методом нормализации, научиться принимать решения о том, какие данные в какие таблицы помещать.

2. Научиться определать отношения между таблицами в базе данных.

3. Научиться использовать механизм транзакций в базе данных.

4. Изучить команду EXPLAIN для анализа эффективности выполнения SQL – запросов.

**Оглавление**

[Проектирование базы данных 3](#__RefHeading___Toc4941_1247080370)

[Первичные ключи: ключи к реляционным базам данных 4](#__RefHeading___Toc4943_1247080370)

[Нормализация 4](#__RefHeading___Toc4945_1247080370)

[Первая нормальная форма 5](#__RefHeading___Toc4947_1247080370)

[Вторая нормальная форма 7](#__RefHeading___Toc4949_1247080370)

[Третья нормальная форма 10](#__RefHeading___Toc4951_1247080370)

[Отношения 12](#__RefHeading___Toc4955_1247080370)

[«Один к одному» 12](#__RefHeading___Toc4957_1247080370)

[«Один ко многим» 13](#__RefHeading___Toc5531_1247080370)

[«Многие ко многим» 14](#__RefHeading___Toc5533_1247080370)

[Базы данных и анонимность 15](#__RefHeading___Toc5535_1247080370)

[Транзакции 15](#__RefHeading___Toc5537_1247080370)

[Ядра (механизмы хранения) транзакций 16](#__RefHeading___Toc7237_2753689920)

[Команда BEGIN 17](#__RefHeading___Toc7239_2753689920)

[Команда COMMIT 17](#__RefHeading___Toc7241_2753689920)

[Команда ROLLBACK 17](#__RefHeading___Toc7243_2753689920)

[Команда EXPLAIN 18](#__RefHeading___Toc7245_2753689920)

[Вопросы 20](#__RefHeading___Toc7263_2753689920)

## Проектирование базы данных

За годы существования баз данных были разработаны руководства, следуя которым можно обеспечить эффективную работу с ними и возможность их масштабирования по мере напол-нения все новыми и новыми данными.

Перед тем как создавать базу данных, очень важно ее удачно спроектировать, в противном случае, скорее всего, придется возвращаться назад и изменять ее структуру, разбивая одни и объединяя другие таблицы и перемещая различные столбцы из таблицы в таблицу с целью достижения рациональных связей, которыми MySQL будет легче воспользоваться.

Для начала было бы неплохо сесть за стол с листом бумаги и карандашом и набросать подборку тех запросов, которые, на ваш взгляд, чаще всего будут нужны вам и вашим пользователям. Для базы данных **книжного интернет-магазина** могут быть записаны следующие вопросы.

* Сколько авторов, книг и покупателей имеется в базе данных?
* Каким автором написана та или иная книга?
* Какие книги написаны тем или иным автором?
* Какая книга продается по самой высокой цене?
* Какая книга является лидером продаж?
* Какие книги не покупались в этом году?
* Какие книги приобретены тем или иным покупателем?
* Какие книги были приобретены одновременно с другими книгами?

Разумеется, к такой базе данных может быть сделано и множество других запросов, но даже эта подборка даст вам представление о том, как следует спланировать структуру таблиц.

Например, книги и номера ISBN должны быть, наверное, скомбинированы в одной таблице, поскольку они тесно взаимосвязаны (некоторые тонкости этого вопроса будут исследованы чуть позже). В отличие от этого книги и покупатели должны находиться в разных таблицах, поскольку они слабо взаимосвязаны. Покупатель может купить любую книгу и даже несколько экземпляров одной и той же книги, а книга может быть приобретена многими

покупателями и может не привлечь внимания еще большего количества потенциальных покупателей.

Когда планируется множество поисковых операций по каким-нибудь столбцам, зачастую их поиск стоит поместить в общую таблицу. А когда какие-то элементы слабо связаны друг с другом, их лучше поместить в отдельные таблицы.

Если принять во внимание эти элементарные правила, то можно предположить, что для удовлетворения всех этих запросов нам понадобятся как минимум три таблицы.

* authors (авторы). Предполагается большое количество поисков по авторам, многие из которых сотрудничали при написании книг, а значит, будут показаны вместе. Оптимальных результатов поиска можно добиться, если о каждом авторе будет дана вся относящаяся к нему информация, следовательно, нам нужна таблица авторов — authors.
* books (книги). Многие книги появляются в различных изданиях. Иногда у них разные издатели, а иногда разные книги имеют одно и то же название. Связи между книгами и авторами настолько сложны, что для книг нужна отдельная таблица.
* customers (покупатели). Причина, по которой покупатели должны находиться в собственной таблице, еще более прозрачна — покупатели могут приобрести любую книгу любого автора.

### Первичные ключи: ключи к реляционным базам данных

Используя возможности реляционных баз данных, мы можем задавать всю информацию для каждого автора, книги и покупателя в одном и том же месте. Очевидно, что нас интересуют связи между ними, например, кто написал каждую книгу и кто ее приобрел, и мы можем сохранить эту информацию лишь за счет создания связей между тремя таблицами. Рассмотрим основные принципы, которые нетрудно усвоить на практике.

Секрет заключается в присваивании каждому автору уникального идентификатора. То же самое делается для каждой книги и каждого покупателя, нам нужен первичный ключ. Для книги имеет смысл использовать в этом качестве номер ISBN, хотя вам, может быть, придет-ся столкнуться с несколькими одинаковыми книгами, имеющими разные номера ISBN. Авторам и покупателям можно просто назначить произвольные ключи, имеющие свойство автоприращения — AUTO\_INCREMENT, что делается весьма просто.

Проще говоря, каждая таблица будет спроектирована вокруг какого-нибудь объекта, в котором, скорее всего, будет вестись интенсивный поиск, — в данном случае вокруг автора, книги или покупателя, и этот объект должен иметь первичный ключ. В качестве ключа не следует выбирать ничего, что могло бы иметь одинаковое значение для разных объектов. Ситуация с номером ISBN является тем самым редким случаем, когда сама издательская индустрия предоставила нам первичный ключ, который можно считать уникальным для каждого продукта. В большинстве случаев для этих целей следует создавать произвольный

ключ, использующий свойство AUTO\_INCREMENT.

## Нормализация

Процесс распределения данных по таблицам и создания первичных ключей называется нормализацией. Основная цель нормализации — обеспечить, чтобы каждая порция информации появлялась в базе данных только один раз. Дублирование данных приводит к крайне неэффективной работе, поскольку неоправданно увеличивает объем базы данных и замедляет тем самым доступ к информации. Еще важнее то, что дубликаты создают большой риск обновления только одной строки продублированных данных и приводят к несогласованности в базе данных, являющейся потенциальным источником серьезных ошибок.

Если, к примеру, названия книг перечисляются и в таблице авторов, и в таблице книг и возникает необходимость исправить опечатку в названии, нужно будет вести поиск в обеих таблицах и вносить одинаковые изменения везде, где встречаются названия книг. Лучше хранить названия в одном месте, а в других местах использовать номер ISBN.

В процессе разбиения базы данных на несколько таблиц важно не зайти слишком далеко и не создать больше таблиц, чем требуется, что может также привести к неэффективности конструкции и замедлению доступа к данным.

К счастью, изобретатель реляционной модели Эдгар Кодд проанализировал понятие нормализации и разбил его на три отдельные схемы, названные первой, второй и третьей нормальными формами. Если вносить изменения в базу данных, последовательно удовлетворяющие требованиям каждой из этих форм, то будет обеспечена оптимальная сбалансированность базы данных, способствующая достижению быстрого доступа и использованию минимального объема оперативной и дисковой памяти.

Чтобы понять, как выполняется нормализация, начнем с весьма несуразной базы

данных, представленной в табл. 9.1, в которой имеется одна таблица, содержащая все сведения об авторах, книгах и вымышленных покупателях. Ее можно рассматривать в качестве первой попытки создания таблицы, отслеживающей, кто из покупателей какие книги заказал. Неэффективность такой конструкции не вызывает сомнений, поскольку данные повсеместно дублируются (дубликаты в таблице выделены полужирным шрифтом), но это всего лишь наша отправная точка.

**Таблица 9.1.** Крайне неэффективная конструкция таблицы базы данных

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Author 1**  **(Автор 1)** | **Author 2**  **(Автор 2)** | **Title**  **(Название)** | **ISBN** | **Price**  **(Цена)** | **Customer**  **name (Имя**  **покупателя)** | **Customer**  **address (Адрес**  **покупателя)** | **Purchase**  **date (Дата**  **покупки)** |
| David  Sklar | Adam  Trachtenberg | PHP Cookbook | 0596101015 | 44,99 | Emma Brown | 1565 Rainbow  Road, Los  Angeles,  CA 90014 | Mar 03  2009 |
| Danny  Goodman |  | Dynamic  HTML | 0596527403 | 59,99 | **Darren**  **Ryder** | **4758 Emily**  **Drive,**  **Richmond,**  **VA 23219** | **Dec 19**  **2008** |
| Hugh E.  Williams | David Lane | PHP and  MySQL | 0596005436 | 44,95 | Earl B.  Thurston | 862 Gregory  Lane,  Frankfort,  KY 40601 | Jun 22  2009 |
| David  Sklar | Adam  Trachtenberg | PHP Cookbook | 0596101015 | 44,99 | **Darren**  **Ryder** | **4758 Emily**  **Drive,**  **Richmond,**  **VA 23219** | **Dec 19**  **2008** |
| Rasmus  Lerdorf | Kevin Tatroe  & Peter  MacIntyre | Programming  PHP | 0596006815 | 39,99 | David Miller | 3647 Cedar  Lane,  Waltham,  MA 02154 | Jan 16  2009 |

В следующих трех разделах мы проанализируем эту конструкцию базы данных,

и вы увидите, как она может быть улучшена за счет удаления продублированных

записей и разбиения одной таблицы на несколько более практичных таблиц,

в каждой из которых будет храниться один тип данных.

### Первая нормальная форма

Чтобы база данных соответствовала первой нормальной форме, она должна выполнять три требования.

1. В ней не должно быть повторяющихся столбцов (атрибутов) с одинаковым смыслом.
2. Пересечение любой строки и столбца должно содержать только одно значение (**все атрибуты атомарны**).
3. Для уникальной идентификации каждой строки должен существовать первичный (главный) ключ (один или более атрибутов, самый малый из возможных).

Рассматривая по порядку эти требования, замечаем, что в столбцы Author 1

и Author 2 заложены повторяющиеся по смыслу данные. Итак, у нас уже появился столбец, который следует поместить в отдельную таблицу, поскольку повторяющийся столбец Author противоречит правилу 1. Второе несоответствие связано с тем, что для последней книги, Programming PHP, указаны три автора. Использование одного и того же столбца Author 2 для имен двух авторов — Kevin Tatroe и Peter MacIntyre — нарушает правило 2. Это еще одна причина перемещения всех сведений об авторах в отдельную таблицу. А вот правило 3 здесь соблюдается, потому что первичный ключ в столбце ISBN уже создан.

В табл. 9.2 показаны результаты перемещения столбцов авторов из табл. 9.1. Теперь здесь уже меньше беспорядка, хотя все еще остаются дубликаты, выделенные полужирным шрифтом.

**Таблица 9.2.** Результаты удаления столбцов Author 1 и Author 2 из табл. 9.1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Title (Название) | ISBN | Price(Цена) | Customer name (Имя покупателя) | Customer address (Адрес покупателя) | Purchase date (Дата покупки) |
| PHP Cookbook | 0596101015 | 44,99 | Emma Brown | 1565 Rainbow Road,Los Angeles, CA 90014 | Mar 03 2009 |
| Dynamic HTML | 0596527403 | 59,99 | **Darren Ryder** | **4758 Emily Drive,Richmond, VA 23219** | **Dec 19 2008** |
| PHP and MySQL | 0596005436 | 44,95 | Earl B. Thurston | 862 Gregory Lane,Frankfort, KY 40601 | Jun 22 2009 |
| PHP Cookbook | 0596101015 | 44,99 | **Darren Ryder** | **4758 Emily Drive,Richmond, VA 23219** | **Dec 19 2008** |
| Programming PHP | 0596006815 | 39,99 | David Miller | 3647 Cedar Lane,Waltham, MA 02154 | Jan 16 2009 |

Новая таблица Authors, приведенная в табл. 9.3, проста по структуре и имеет довольно небольшой размер. В ней просто перечисляются номера ISBN, принадлежащие книге с тем или иным названием, рядом с которыми размещается фамилия автора. Если у книги более одного автора, соавторы получают собственную строку. Поначалу эта таблица может показаться несуразной, потому что по ней нельзя понять сразу, кто из авторов какую книгу написал. Но не стоит переживать: MySQL может быстро проинформировать вас об этом. Для

этого нужно лишь сообщить, для какой именно книги нужна такая информация, и MySQL воспользуется ее ISBN для поиска в таблице авторов, что займет какие-то миллисекунды.

**Таблица 9.3.** Новая таблица Authors

| **ISBN** | **Author (Автор)** |
| --- | --- |
| 0596101015 | David Sklar |
| 0596101015 | Adam Trachtenberg |
| 0596527403 | Danny Goodman |
| 0596005436 | Hugh E. Williams |
| 0596005436 | David Lane |
| 0596006815 | Rasmus Lerdorf |
| 0596006815 | Kevin Tatroe |
| 0596006815 | Peter MacIntyre |

Как было отмечено ранее, ISBN будет служить в качестве первичного ключа для таблицы книг — Books, когда дело дойдет до ее создания. Я упомянул об этом, чтобы подчеркнуть, что ISBN, тем не менее, не является первичным ключом для таблицы Authors. При практической разработке для таблицы Authors также нужно создать первичный ключ, обеспечивающий уникальную идентификацию авторов.

Поэтому для таблицы Authors атрибут ISBN является просто атрибутом, для которого в целях ускорения поиска может быть создан ключ, но этот ключ будет уже не первичным. Фактически в этой таблице он и не может быть первичным, поскольку не обладает уникальностью: один и тот же номер ISBN появляется по нескольку раз в тех случаях, когда над одной книгой работали несколько авторов.

Поскольку мы будем использовать такой ключ для связи авторов с книгами в другой таблице, этот атрибут называется *внешним* ключом.

Ключи (которые также называются индексами) имеют в MySQL несколько предназначений. Основной целью создания ключа является ускорение поиска (в условиях WHERE sql запросов). Но ключ можно применять и для уникальной идентификации элемента. Таким образом, уникальный ключ часто задействуется в качестве первичного ключа в одной таблице и в качестве внешнего ключа для связи строк этой таблицы со строками другой.

### Вторая нормальная форма

Первая нормальная форма позволяет разобраться с продублированными данными (или избыточностью) в нескольких столбцах. Вторая нормальная форма имеет отношение только к решению проблемы избыточности в нескольких строках. Чтобы привести базу данных ко второй нормальной форме, ваши таблицы должны уже иметь первую нормальную форму. Как только это будет сделано, для определения столбцов, данные в которых повторяются в разных

местах, и последующего их перемещения в собственные таблицы применяется вторая нормальная форма. Второе правило нормализации требует, чтобы **любой неключевой столбец зависел от всего первичного ключа**. Следовательно, таблица не должна содержать неключевых столбцов, зависящих только от части составного первичного ключа.

Еще раз посмотрим на табл. 9.2. Видите, Darren Ryder приобрел две книги, и поэтому его данные продублированы. Это говорит о том, что столбцы, имеющие отношение к покупателю (Customer name и Customer address), следует переместить в их собственные таблицы. В табл. 9.4 показан результат удаления двух столбцов, касающихся покупателя, из табл. 9.2.

**Таблица 9.4.** Новая таблица Titles

| **ISBN** | **Title (Название)** | **Price (Цена)** |
| --- | --- | --- |
| 0596101015 | PHP Cookbook | 44,99 |
| 0596527403 | Dynamic HTML | 59,99 |
| 0596005436 | PHP and MySQL | 44,95 |
| 0596006815 | Programming PHP | 39,99 |

Таким образом, в табл. 9.4 остались только столбцы номера ISBN, названия (Title) и цены (Price) для четырех уникальных книг, поэтому теперь это эффективная в использовании и независимая таблица, удовлетворяющая требованиям как первой, так и второй нормальной формы. Попутно мы справились с сокращением информации до уровня тех данных, которые имеют непосредственное отношение к книгам с определенными названиями. Эта таблица может также включать год издания, количество страниц, количество переизданий и т. д., поскольку все эти данные имеют тесную связь друг с другом. Единственное правило гласит:

сюда нельзя помещать столбцы, которые могут содержать несколько значений для одной книги, поскольку тогда нам придется указывать одну и ту же книгу в нескольких строках, нарушая таким образом правила второй нормальной формы. К примеру, к нарушениям на этом этапе нормализации может привести восстановление столбца авторов.

Но изучая извлеченные столбцы, относящиеся к покупателям, которые теперь показаны в табл. 9.5, можно заметить, что эта таблица все же требует дополнительной нормализации, поскольку сведения о покупателе Darren Ryder по-прежнему продублированы. Следует также признать, что правило 2 первой нормальной формы (все столбцы должны содержать только одно значение) здесь не соблюдается, поскольку адресные данные нужно разбить на отдель-

ные столбцы для адреса — Address, города — City, штата — State и почтового индекса — Zip.

**Таблица 9.5.** Сведения о покупателях из табл. 9.2

| **ISBN** | **Customer name (Имя покупателя)** | **Customer address (Адрес покупателя)** | **Purchase date (Дата покупки)** |
| --- | --- | --- | --- |
| 0596101015 | Emma Brown | 1565 Rainbow Road, Los Angeles, CA 90014 | Mar 03 2009 |
| 0596527403 | **Darren Ryder** | **4758 Emily Drive, Richmond, VA 23219** | **Dec 19 2008** |
| 0596005436 | Earl B. Thurston | 862 Gregory Lane, Frankfort, KY 40601 | Jun 22 2009 |
| 0596101015 | **Darren Ryder** | **4758 Emily Drive, Richmond, VA 23219** | **Dec 19 2008** |
| 0596006815 | David Miller | 3647 Cedar Lane, Waltham, MA 02154 | Jan 16 2009 |

Нужно продолжить разбиение этой таблицы, чтобы обеспечить однократный ввод каждого из сведений, касающихся покупателя. Поскольку ISBN не относится к таким сведениям и не может использоваться в качестве первичного ключа для идентификации покупателей (или авторов), должен быть создан новый ключ.

В табл. 9.6 показан результат нормализации таблицы Customers в соответствии с правилами первой и второй нормальных форм. Теперь у каждого покупателя есть уникальный номер покупателя, который называется CustNo, используется в качестве первичного ключа и который, скорее всего, был создан с использованием свойства автоприращения — AUTO\_INCREMENT. Все составляющие адресов были также распределены по разным столбцам, для того чтобы упростить их поиск и обновление.

В то же время для нормализации табл. 9.6 необходимо было удалить информацию о покупках, поскольку в противном случае в ней встречались бы одни и те же сведения о покупателе для каждой купленной им книги. Вместо этого данные о покупках теперь помещены в новую таблицу Purchases (табл. 9.7).

**Таблица 9.6.** Новая таблица Customers (покупатели)

| **CustNo (Номер покупателя)** | **Name (Имя)** | **Address (Адрес)** | **City (Город)** | **State (Штат)** | **Zip (Почтовый индекс)** |
| --- | --- | --- | --- | --- | --- |
| 1 | Emma Brown | 1565 Rainbow Road | Los Angeles | CA | 90014 |
| 2 | Darren Ryder | 4758 Emily Drive | Richmond | VA | 23219 |
| 3 | Earl B. Thurston | 862 Gregory Lane | Frankfort | KY | 40601 |
| 4 | David Miller | 3647 Cedar Lane | Waltham | MA | 02154 |

**Таблица 9.7.** Новая таблица Purchases (покупки)

| **CustNo (Номер покупателя)** | **ISBN** | **Date (Дата)** |
| --- | --- | --- |
| 1 | 0596101015 | Mar 03 2009 |
| 2 | 0596527403 | Dec 19 2008 |
| 2 | 0596101015 | Dec 19 2008 |
| 3 | 0596005436 | Jun 22 2009 |
| 4 | 0596006815 | Jan 16 2009 |

Здесь в качестве ключа, связывающего вместе таблицы Customers и Purchases, опять исполь-зуется атрибут CustNo из табл. 9.6. Поскольку здесь повторно появляется атрибут ISBN, эта таблица может быть связана также с таблицами Authors и Titles.

Атрибут CustNo может быть полезна в качестве ключа (но только не первичного) в таблице Purchases: один и тот же покупатель может приобрести несколько книг (и даже несколько экземпляров одной и той же книги), поэтому отрибут CustNo не может служить первичным ключом. Фактически у таблицы Purchases вообще нет первичного ключа. И это вполне нормально, поскольку потребностей в от слеживании уникальных покупок не предвидится. Если один покупатель приобретет два экземпляра одной и той же книги, то придется смириться с двумя строками, содержащими одну и ту же информацию. Для упрощения поиска можно определить в качестве ключей, только не первичных, обе столбцы, CustNo и ISBN.

Теперь у нас четыре таблицы, на одну больше, чем те три, которые потребовались бы согласно нашим первоначальным прикидкам. Мы пришли к этому решению в процессе нормализации, методически следуя правилам первой и второй нормальных форм, которые однозначно позволили выявить необходимость существования дополнительной, четвертой таблицы под названием Purchases (Покупки).

У нас есть следующие таблицы: Authors (см. табл. 9.3), Titles (см. табл. 9.4), Customers (см. табл. 9.6) и Purchases (см. табл. 9.7). Каждая из них может быть связана с любой другой с помощью либо ключа CustNo, либо ключа ISBN.

Например, чтобы посмотреть, какие книги приобрел Darren Ryder, их можно поискать через табл. 9.6, Customers, где мы увидим, что CustNo этого покупателя 2. Теперь, имея этот номер, можно перейти к табл. 9.7, Purchases, найти там графу ISBN и увидеть, что он приобрел книги с номерами 0596527403 и 0596101015 19 декабря 2008 года. Подобные поиски кажутся утомительными для человека, но не составляют ни малейшего труда для MySQL.

Определить названия этих книг можно, обратившись затем к табл. 9.4, Titles, и увидев, что это книги Dynamic HTML и PHP Cookbook. Если нужно узнать авторов этих книг, следует воспользоваться номерами ISBN, которые теперь нужно найти в табл. 9.3, Authors. Станет понятно, что книгу с номером ISBN 0596527403, Dynamic HTML, написал Danny Goodman, а авторы книги с номером ISBN 0596101015, PHP Cookbook — David Sklar и Adam Trachtenberg.

### Третья нормальная форма

После приведения в соответствие первой и второй нормальным формам база данных приоб-рела подходящий вид, и в дальнейшем вам, возможно, уже не придется что-либо в ней изменять. Но если применить к базе данных более суровые требования, то можно довести ее до соответствия правилам третьей нормальной формы, которые требуют, **чтобы данные, не имеющие непосредственной зависимости от первичного ключа, но имеющие зависи-мость от другого значения в таблице, были также перемещены в отдельные таблицы в соответствии с тем, к чему они имеют отношение**.

Например, касательно табл. 9.6, Customers, можно утверждать, что ключи State, City и Zip не имеют прямого отношения к каждому покупателю, поскольку эти же составляющие будут присутствовать в адресах многих других людей. Но они напрямую связаны друг с другом тем, что улица в адресе — Address относится к городу — City, а город относится к штату — State.

Поэтому чтобы соблюсти правила третьей нормальной формы для табл. 9.6, ее нужно разбить на табл. 9.8–9.11.

Ну и как пользоваться этим набором из четырех таблиц вместо одной табл. 9.6? В табл. 9.8 нужно найти Zip-код, затем в табл. 9.9 — соответствующий ему город. Располагая этой информацией, в табл. 9.10 можно найти название города, а затем — идентификатор штата — StatelD, который можно использовать в табл. 9.11 для поиска его названия.

Хотя подобное подстраивание под третью нормальную форму может показаться излишним, у него могут быть и свои преимущества. Например, взгляните на табл. 9.11, в которую удалось включить как название, так и двухбуквенную аббревиатуру штата. Сюда же при желании можно также включить данные о количестве жителей и другие демографические сведения.

**Таблица 9.8.** Таблица Customers, соответствующая правилам третьей нормальной формы

| **CustNo (Номер покупателя)** | **Name (Имя)** | **Address (Адрес)** | **Zip (Почтовый индекс)** |
| --- | --- | --- | --- |
| 1 | Emma Brown | 1565 Rainbow Road | 90014 |
| 2 | Darren Ryder | 4758 Emily Drive | 23219 |
| 3 | Earl B. Thurston | 862 Gregory Lane | 40601 |
| 4 | David Miller | 3647 Cedar Lane | 02154 |

**Таблица 9.9.** Таблица с индексами городов, соответствующая правилам третьей нормальной формы

| **Zip (Почтовый индекс)** | **CitylD (Идентификатор города)** |
| --- | --- |
| 90014 | 1234 |
| 23219 | 5678 |
| 40601 | 4321 |
| 02154 | 8765 |

**Таблица 9.10.** Таблица Cities, соответствующая правилам третьей нормальной формы

| **CityID (Идентификатор города)** | **Name (Название)** | **StateID (Идентификатор штата)** |
| --- | --- | --- |
| 1234 | Los Angeles | 5 |
| 5678 | Richmond | 46 |
| 4321 | Frankfort | 17 |
| 8765 | Waltham | 21 |

**Таблица 9.11.** Таблица States, соответствующая правилам третьей нормальной формы

| **StateID (Идентификатор штата)** | **Name (Название)** | **Abbreviation (Аббревиатура)** |
| --- | --- | --- |
| 5 | California | CA |
| 46 | Virginia | VA |
| 17 | Kentucky | KY |
| 21 | Massachusetts | MA |

Таблица 9.10 может также содержать более локализованную демографическую информацию, которая может оказаться полезной вам и (или) вашим покупателям. Разбивая эти данные на части, вы можете упростить обслуживание своей базы данных в будущем, когда потребуется добавить к таблицам дополнительные столбцы.

Решение о том, к чему именно следует применить правило третьей нормальной формы, может оказаться непростым. Оценка должна основываться на том, какие дополнительные данные могут понадобиться со временем. Если вы абсолютно уверены в том, что ничего, кроме имени и адреса покупателя, не понадобится, то, наверное, без этой заключительной стадии нормализации можно будет обойтись.

С другой стороны, представьте, что вы создаете базу данных для такой крупной организации, как Почтовая служба страны. Что вы будете делать, если город будет переименован? С такой таблицей, как табл. 9.6, вам придется проводить глобальный поиск и менять название города везде, где оно упоминается. Но если ваша база данных нормализована по правилам третьей нормальной формы, нужно будет изменить всего лишь одну запись в табл. 9.10 для того, чтобы это изменение отразилось на всей базе данных.

Поэтому необходимо ответить себе на два вопроса, которые помогут принять решение, нужно ли применять нормализацию по правилам третьей нормальной формы к той или иной таблице.

* Существует ли вероятность того, что к таблице нужно будет добавить много новых столбцов?
* Может ли когда-нибудь для любого из полей этих таблиц потребоваться глобальное обновление?

Если оба ответа на эти вопросы положительные, наверное, все же следует провести заключительную стадию нормализации.

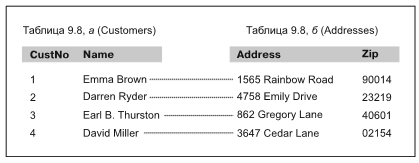
## Отношения

MySQL называют системой управления реляционными базами данных, потому что в ее таблицах содержатся не только данные, но и отношения между ними. Существует три категории отношений.

### «Один к одному»

*Отношение «один к одному»* между двумя типами данных похоже на традиционные брачные отношения: каждый элемент данных соотносится только с одним элементом другого типа. Это на удивление редкий тип отношений. Например, автор может написать несколько книг, у книги может быть несколько авторов, и даже адрес может быть связан с несколькими покупателями. Возможно, наилучшим примером, встречавшимся в этой работе, может послужить отношение «один к одному» между названием штата и его двухбуквенной аббревиатурой.

Чтобы легче было объяснить, что это такое, предположим, что по какому-нибудь конкретному адресу может проживать только один покупатель. В таком случае отношение Customers — Addresses на рис. 9.1 будет отношением «один к одному»: только один покупатель живет по каждому адресу, и по каждому адресу может жить только один покупатель.



**Рис. 9.1.** Таблица покупателей, Customers (табл. 9.8), разбитая на две таблицы

Обычно когда у двух элементов имеется отношение «один к одному», их включают в качестве столбцов в одну и ту же таблицу. Для отнесения их к двум отдельным таблицам могут быть две причины:

* вы хотите быть готовыми к тому, что позже это отношение изменится и больше уже не будет иметь характера «один к одному»;
* в таблице слишком много столбцов, и вы полагаете, что производительность работы системы или возможности ее обслуживания улучшатся за счет ее разбиения.

Разумеется, когда дело дойдет до создания настоящей базы данных, между покупателями и адресами нужно будет создать отношения «один ко многим» (*один* адрес, много покупателей).

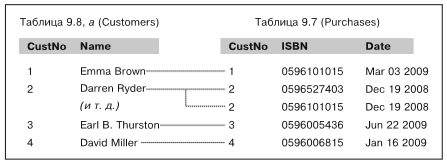
### «Один ко многим»

Отношения «один ко многим» (или «многие к одному») возникают в том случае, когда одна строка в одной таблице связана со многими строками в другой таблице. Вы уже поняли, что в табл. 9.8 возникли бы отношения «один ко многим», если бы несколько покупателей проживали по одному и тому же адресу. В таком случае ее нужно разбить.

Если посмотреть на табл. 9.8, а, показанную на рис. 9.1, можно увидеть, что у нее имеется отношение «один ко многим» с таблицей покупок, табл. 9.7, поскольку каждый покупатель представлен только одним конкретным человеком из табл. 9.8, а.

Но табл. 9.7, Purchases, может содержать (и содержит) более одной покупки, сделанной одним и тем же покупателем. Поэтому один покупатель имеет от ношение ко *многим* покупкам.

На рис. 9.2 эти две таблицы показаны рядом друг с другом, а линии соединяют строки в каждой таблице и, начинаясь в одной строке левой таблицы, могут соединять с ней более одной строки правой таблицы. Схема отношения «один ко многим» также хорошо подходит и для описания отношения «многие к одному», в этом случае нужно левую и правую таблицы поменять местами и рассматривать их как отношение «один ко многим».



**Рис. 9.2.** Иллюстрация отношения между двумя таблицами

Чтобы отобразить в реляционной базе данных отношение «один ко многим», создайте таблицу для «многих» и таблицу для «одного». Таблица для «многих» должна содержать столбец с перечислением первичного ключа из таблицы для «одного». Таким образом, таблица Purchases (Покупки) будет содержать столбец с перечислением первичного ключа покупателя.

### «Многие ко многим»

В отношении «многие ко многим» многие строки в одной таблице связаны со многими строками в другой таблице. Чтобы создать это отношение, нужно добавить третью таблицу, содержащую по столбцу из каждой из этих двух таблиц. В третьей таблице больше ничего не содержится, она предназначена только для связи других таблиц.

Именно такой промежуточной таблицей и является табл. 9.12. Она была извлечена из табл. 9.7, Purchases (Покупки), но в ней отсутствует информация о дате покупки. Теперь она содержит копию номера ISBN каждой проданной книги, а также номер покупателя.

С помощью этой промежуточной таблицы можно пройти по всем хранящимся в базе данным, пользуясь схемой их отношений. За отправную точку можно взять адрес и найти авторов любых книг, приобретенных покупателем, проживающим по этому адресу.

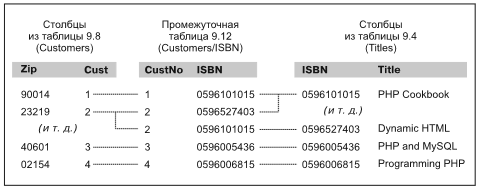
**Таблица 9.12.** Промежуточная таблица

| **Customer (Покупатель)** | **ISBN** |
| --- | --- |
| 1 | 0596101015 |
| 2 | 0596527403 |
| 2 | 0596101015 |
| 3 | 0596005436 |
| 4 | 0596006815 |

Предположим, к примеру, что нужно найти покупки, связанные с почтовым индексом 23219. Если поискать этот почтовый индекс в табл. 9.8, б, то можно обнаружить, что покупатель с номером 2 приобрел как минимум одну книгу, имеющуюся в базе данных. Теперь можно воспользоваться табл. 9.8, а и найти имя этого покупателя или воспользоваться новой промежуточной табл. 9.12, для того чтобы найти приобретенную им книгу или книги.

По этой таблице можно определить, что были приобретены две книги, и, отследив их номера в табл. 9.4, найти названия и цены этих книг или обратиться к табл. 9.3 и увидеть в ней их авторов.

Если вам показалось, что все это, по сути, не что иное, как сочетание нескольких отношений «один ко многим», то так оно и есть. Чтобы проиллюстрировать это, на рис. 9.3 все три таблицы представлены вместе.



**Рис. 9.3.** Создание отношения «многие ко многим» с помощью третьей таблицы

Проследите по любому почтовому индексу (zip-коду) в левой таблице связанные с ним идентификаторы покупателей. Далее можно проследить их связь с промежуточной таблицей, которая объединяет левую и правую таблицы путем связывания покупательских идентификаторов и номеров ISBN. Теперь остается только проследовать по ISBN к правой таблице, чтобы увидеть, к какой книге он относится.

Промежуточную таблицу можно использовать также для следования в обратном направлении — от названий книг до zip-кода. Из таблицы Titles можно взять ISBN, которым воспользоваться для поиска в промежуточной таблице идентификационных номеров покупателей этих книг, и, наконец, в таблице Customers идентификационные номера будут сопоставлены с zip-кодами мест проживания покупателей.

## Базы данных и анонимность

Интересный аспект использования отношений заключается в том, что о каком-нибудь элементе, например покупателе, можно собрать массу сведений, не зная ничего о его личности. Обратите внимание на то, что в предыдущем примере мы прошли от покупательских zip-кодов к их покупкам и вернулись назад, не определяя имен покупателей. Базы данных могут использоваться не только для отслеживания сведений о людях, но и для защиты относящихся к ним конфиденциальных данных, при этом сохраняется возможность поиска полезной информации, например, передача информации о покупке без раскрытия других данных клиента.

## Транзакции

В некоторых приложениях жизненно необходимо, чтобы последовательность запросов шла в нужном порядке и при этом каждый отдельный запрос успешно завершался. Представим, например, что создается последовательность запросов для перевода средств с одного банковского счета на другой. Вам бы не хотелось, чтобы при этом происходило что-либо подобное:

* вы зачислили средства на второй счет, а когда попытались снять их с первого счета, при обновлении данных произошел сбой, и теперь эти средства числятся на обоих счетах;
* вы сняли средства с первого банковского счета, но при запросе на обновление с целью их зачисления на второй счет произошел сбой, и теперь эти средства бесследно исчезли.

Как видите, для этого типа транзакций важен не только порядок выполнения запросов, необходимо также, чтобы все части транзакции завершились успешно. Но как все это обеспечить? Ведь после осуществления запроса аннулировать его уже невозможно. Необходимо ли отслеживать все части транзакции, а затем проводить полный откат, если одна из ее частей даст сбой? Ничего этого делать не нужно, поскольку MySQL поставляется с мощным средством обработки транзакций, которое защищает именно от таких непредвиден-ных обстоятельств.

Кроме того, транзакции предоставляют одновременный доступ к базе данных множеству пользователей или программ за счет обеспечения очередности проведения всех транзакций, и каждый пользователь или программа соблюдают очередность, не наступая друг другу на пятки, — MySQL со всем этим прекрасно справляется.

### Ядра (механизмы хранения) транзакций

Чтобы использовать имеющееся в MySQL средство обработки транзакций, нужно задействовать MySQL-ядро InnoDB (используемое по умолчанию, начиная с версии 5.5). Если вы не уверены в номере используемой версии MySQL, на которой будет запускаться ваш код, нужно не выстраивать предположения, что InnoDB является механизмом по умолчанию, а использовать его принудительно при создании таблицы, как показано далее.

Создадим таблицу банковских счетов, введя команды, показанные в примере 9.1.

(Напомню, что для этого вам нужно получить доступ к командной строке MySQL

и воспользоваться подходящей для этой таблицы базой данных.)

**Пример 9.1.** Создание таблицы, готовой к обработке транзакций

CREATE TABLE accounts (

number INT, balance FLOAT, PRIMARY KEY(number)

) ENGINE InnoDB;

DESCRIBE accounts;

Команда, которая находится в последней строке этого примера, отобразит структуру новой таблицы, позволяя убедиться в ее успешном создании. Будет выведена следующая информация:

+---------+---------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+---------+---------+------+-----+---------+-------+

| number | int(11) | NO | PRI | NULL | |

| balance | float | YES | | NULL | |

+---------+---------+------+-----+---------+-------+

2 rows in set (0.00 sec)

Теперь создадим в этой таблице две строки, которые можно будет задействовать в транзакциях. Для этого введем команды, показанные в примере 9.2.

**Пример 9.2.** Заполнение таблицы accounts

INSERT INTO accounts(number, balance) VALUES(12345, 1025.50);

INSERT INTO accounts(number, balance) VALUES(67890, 140.00);

SELECT \* FROM accounts;

Команда в третьей строке отобразит содержимое таблицы, подтверждая успешное создание строк. Будет выведена следующая информация:

+--------+---------+

| number | balance |

+--------+---------+

| 12345 | 1025.5 |

| 67890 | 140 |

+--------+---------+

2 rows in set (0.00 sec)

После создания и предварительного заполнения этой таблицы можно приступить к использованию транзакций.

### Команда BEGIN

Транзакции в MySQL начинаются либо с команды BEGIN, либо с команды START TRANSACTION. Чтобы отправить транзакцию системе MySQL, введите команды, показанные в примере 9.3.

**Пример 9.3.** Транзакция MySQL

BEGIN;

UPDATE accounts SET balance=balance+25.11 WHERE number=12345;

COMMIT;

SELECT \* FROM accounts;

Результаты этой транзакции выводятся командой, содержащейся в последней строке, и должны иметь следующий вид:

+--------+---------+

| number | balance |

+--------+---------+

| 12345 | 1050.61 |

| 67890 | 140 |

+--------+---------+

2 rows in set (0.00 sec)

Как видите, баланс счета 12345 увеличился на 25,11 и теперь составляет 1050,61.

В примере 9.3 можно было также заметить команду COMMIT, которая рассматривается в следующем разделе.

### Команда COMMIT

Когда вы убедитесь в том, что ряд запросов, входящих в транзакцию, успешно выполнен, введите команду COMMIT, чтобы передать все изменения базе данных. До тех пор, пока не будет получена команда COMMIT, все внесенные изменения рассматриваются MySQL как временные. Эта особенность позволяет отменить транзакцию, отправляя вместо команды передачи COMMIT команду отката ROLLBACK.

### Команда ROLLBACK

Используя команду ROLLBACK, можно заставить MySQL забыть обо всех запросах,

выданных с начала транзакции, и отменить транзакцию. Можете проверить эту команду в действии путем ввода транзакции по переводу средств, показанной в примере 9.4.

**Пример 9.4.** Транзакция по переводу средств

BEGIN;

UPDATE accounts SET balance=balance-250 WHERE number=12345;

UPDATE accounts SET balance=balance+250 WHERE number=67890;

SELECT \* FROM accounts;

Как только будут введены эти строки, вы увидите следующий результат:

+--------+---------+

| number | balance |

+--------+---------+

| 12345 | 800.61 |

| 67890 | 390 |

+--------+---------+

2 rows in set (0.00 sec)

Теперь у первого банковского счета значение на 250 единиц меньше, чем раньше, а значение второго увеличилось на 250 единиц — вы осуществили между ними перевод 250 единиц. А теперь предположим, что что-то пошло не так и эту транзакцию нужно отменить. Для этого нужно лишь ввести команду, показанную в примере 9.5.

**Пример 9.5.** Отмена транзакции с помощью команды ROLLBACK

ROLLBACK;

SELECT \* FROM accounts;

Теперь вы должны увидеть следующую выходную информацию, показывающую восстановление прежнего баланса на обоих счетах, благодаря тому что транзакция была отменена командой ROLLBACK:

+--------+---------+

| number | balance |

+--------+---------+

| 12345 | 1050.61 |

| 67890 | 140 |

+--------+---------+

2 rows in set (0.00 sec)

## Команда EXPLAIN

Система MySQL поставляется с мощным инструментарием, который позволяет исследовать, как она интерпретировала выданные ей запросы. Используя команду EXPLAIN, можно получить отображение состояния любого запроса, чтобы понять, можно ли его выдать более удобным или эффективным способом. Применение этой команды с созданной ранее таблицей accounts показано в примере 9.6.

**Пример 9.6.** Использование команды EXPLAIN

EXPLAIN SELECT \* FROM accounts WHERE number='12345';

Результаты выполнения команды EXPLAIN будут выглядеть следующим образом:

+--+------+--------+------+-----+--------+-------+----+-----+----+------+-----+

|id|select|table |part- |type |possible|key |key |ref |rows|fil- |Extra|

| |\_type | |itions| |\_keys | |\_len| | |tered | |

+--+------+--------+------+-----+--------+-------+----+-----+----+------+-----+

|1 |SIMPLE|accounts|NULL |const|PRIMARY |PRIMARY|4 |const|1 |100.00|NULL |

+--+------+--------+------+-----+--------+-------+----+-----+----+------+-----+

1 row in set (0.00 sec)

Здесь MySQL предоставляет вам следующую информацию.

* select\_type. Тип выборки простой — SIMPLE. При объединении таблиц будет показан объединенный (join) тип.
* table. Текущей запрашиваемой таблицей была accounts.
* type. Тип запроса — const. Если идти от наименее эффективного типа к наиболее эффективному, то возможные значения выстраиваются в следующий ряд: ALL, index, range, ref, eq\_ref, const, system и NULL.
* possible\_keys. Возможно, это первичный ключ, PRIMARY, а это значит, что доступ должен быть быстрым.
* key. В данном случае используется ключ PRIMARY, что является хорошим показа-телем.
* key\_len. Длина ключа равна 4. Это количество байтов индекса, которое будет исполь-зовано MySQL.
* ref. Столбец ref отображает, какие столбцы или константы используются

с ключом. В данном случае применяется константный ключ.

* rows. Количество строк, которые должны быть просмотрены этим запросом, равно 1, что также является хорошим показателем.

Когда появится запрос, который подозревается в лишней трате времени на свое выполнение, попробуйте воспользоваться командой EXPLAIN, чтобы посмотреть, как его можно оптимизировать. Вы сможете обнаружить, какие ключи (если таковые имеются) были задействованы, какова их длина и т. д., и тогда можно будет соответствующим образом подкорректировать запрос или конструкцию таблицы (или таблиц).

После того как эксперименты с временной таблицей accounts будут завершены, может появиться желание удалить эту таблицу с помощью следующей команды:

DROP TABLE accounts;

## Вопросы

1. Что означает слово отношение (relationship) применительно к реляционным базам данных?

2. Какое понятие применяется к процессу удаления повторяющихся данных и оптимизации таблиц?

3. Как формулируются три правила первой нормальной формы?

4. Как привести таблицу в соответствие с правилом второй нормальной формы?

5. Что нужно поместить в столбец, для того чтобы связать две таблицы, содержащие элементы, имеющие отношение «один ко многим»?

6. Как создать базу данных с отношением «многие ко многим»?

7. Какие команды инициируют и завершают транзакцию MySQL?

8. Какие возможности предоставляет MySQL для изучения подробностей работы запроса?