Как важно быть нормальным

До настоящего момента мы не особо задумывались при создании таблиц. Работают — и ладно; в конце концов, с ними можно выполнять команды `SELECT`, `INSERT`, `DELETE` и `UPDATE`. Но при увеличении объема данных постепенно становится ясно, что следовало бы сделать при создании таблицы для упрощения условий `WHERE`: ее следовало бы сделать более нормальной.

Две таблицы

Джек и Марк создали таблицы для хранения информации о рекордах рыбной ловли. В таблице Марка имеются столбцы для бытового и научного названия рыбы, ее веса и места, где она была поймана. Столбца для имени человека, поймавшего рыбу, в этой таблице нет.

Таблица состоит из четырех столбцов. Сравните с таблицей fish_records на следующей странице.

fish_info

common	species	location	weight
большеротый окунь	M. salmoides	Монтгомери Лейк, GA	22 фт 4 унц
судак	S. vitreus	Олд Хикори Лейк, TN	25 фт 0 унц
лосось Кларка	O. Clarki	Пирамид Лейк, NV	41 фт 0 унц
желтый окунь	P. Flavescens	Бордентаун, NJ	4 фт 3 унц
синежаберник	L. Macrochirus	Кетона Лейк, AL	4 фт 12 унц
панцирник	L. Osseus	Тринити Ривер, TX	50 фт 5 унц
белый краппи	P. annularis	Дамба Энид, MS	5 фт 3 унц
красноперая щука	E. americanus	Дьюарт Лейк, IN	1 фт 0 унц
серебристый карась	C. auratus	Лейк Ходжес, CA	6 фт 10 унц
чавыча	O. Tshawytscha	Кенай Ривер, AK	97 фт 4 унц

В таблице Джека тоже хранятся бытовые и научные названия рыб, но в ней также имеются столбцы для имени и фамилии рыболова, а место вылова разбито на два столбца: название водоема хранится отдельно от штата.

Эта таблица тоже содержит информацию о рыболовных рекордах, но в ней почти вдвое больше столбцов.

fish_records

first_name	last_name	common	location	state	weight	date
Джордж	Перри	большеротый окунь	Монтгомери Лейк	GA	22 фт 4 унц	2/6/1932
Мабри	Харпер	судак	Олд Хикори Лейк	TN	25 фт 0 унц	2/8/1960
Джон	Скиммерхорн	лосось Кларка	Пирамид Лейк	NV	41 фт 0 унц	1/12/1925
С.С.	Эббот	желтый окунь	Бордентаун	NJ	4 фт 3 унц	1/5/1865
Т.С.	Хадсон	синежаберник	Кетона Лейк	AL	4 фт 12 унц	9/4/1950
Таунсенд	Миллер	панцирник	Тринити Ривер	TX	50 фт 5 унц	30/7/1954
Фред	Брайт	белый краппи	Дамба Энид	MS	5 фт 3 унц	31/7/1957
Майк	Берг	красноперая щука	Дьюарт Лейк	IN	1 фт 0 унц	9/6/1990
Флорентино	Абена	серебристый карась	Лейк Ходжес	CA	6 фт 10 унц	17/4/1996
Лес	Андерсон	чавыча	Кенай Ривер	AK	97 фт 4 унц	17/5/1985

Возьми в руку карандаш

Напишите запрос для каждой таблицы, возвращающий все записи для штата Нью-Джерси.

Часто Задаваемые Вопросы

В: Выходит, таблица Джека лучше, чем таблица Марка?

О: Нет. Это разные таблицы с разными целями. Марку редко приходится проводить поиск по штату, потому что его интересуют только названия (бытовое и научное) выловленных рыб и их вес.

С другой стороны, Джеку *потребуется* искать данные по штату в своих запросах. Именно поэтому он создал в своей таблице отдельный столбец, чтобы было удобнее указывать штат в запросах.

В: Следует ли избегать оператора `LIKE` в запросах? Что в нем плохого?

О: В операторе `LIKE` нет ничего плохого, но он усложняет структуру запроса и повышает риск получения посторонних результатов. Если столбцы содержат сложную информацию, `LIKE` не позволяет легко и однозначно определить критерий поиска.

В: Почему короткие запросы лучше длинных?

О: Чем проще запрос, тем лучше. С увеличением объема базы данных и добавлением новых таблиц запросы усложняются. Начинайте с самых простых запросов, позднее вы их оцените.

В: Значит, в моих столбцах всегда должны храниться как можно меньшие фрагменты данных?

О: Не обязательно. Как показывает пример с таблицами Марка и Джека, все зависит от *использования* данных. Для примера представьте таблицы со списком машин, предназначенные для автомеханика и продавца. Механику необходима подробная информация о каждой машине, а продавцу может быть достаточно фирмы-производителя, модели и номера.

В: Допустим, в записи хранится почтовый адрес. Почему бы не создать один столбец для хранения полного адреса и несколько других столбцов для хранения его составных частей?

О: Дублирование данных поначалу может показаться вполне разумной мерой, но подумайте, сколько лишнего пространства будет расходоваться на жестком диске, если база данных вырастет до значительных размеров. А еще при дублировании данных в команду `UPDATE` должно включаться дополнительное лишнее условие, и вы должны помнить о нем при каждом изменении данных.

Давайте более подробно разберемся в том, как спроектировать оптимальную структуру таблицы для ваших целей..

Структура таблицы зависит от того, как вы собираетесь использовать свои данные.

МОЗГОВОЙ ШТУРМ

SQL — язык, используемый реляционными базами данных. Как вы думаете, что означает термин «реляционный» в контексте баз данных SQL?

Логические связи как суть таблицы

SQL известен как язык Реляционных Систем Управления Базами Данных (РСУБД). Термин запоминать не обязательно, нас интересует только слово «РЕЛЯЦИОННЫХ*». Для нас оно означает, прежде всего, одно: чтобы правильно спроектировать таблицу, необходимо продумать, как столбцы связываются друг с другом для описания некоторого объекта.

Ваша задача — описать объект при помощи столбцов так, чтобы по возможности упростить получение необходимой информации. Конечно, выбор во многом зависит от ваших требований к таблице, но существуют некоторые общие меры, которые следует принять при выборе структуры таблицы.

1. Выберите один объект, который должна описывать таблица.

Какой основной объект описывает ваша таблица?

2. Составьте список того, что необходимо знать об этом объекте при работе с таблицей.

Как будет использоваться ваша таблица?

3. Используя список, разбейте необходимую информацию об объекте на фрагменты, которые могут использоваться для определения структуры таблицы.

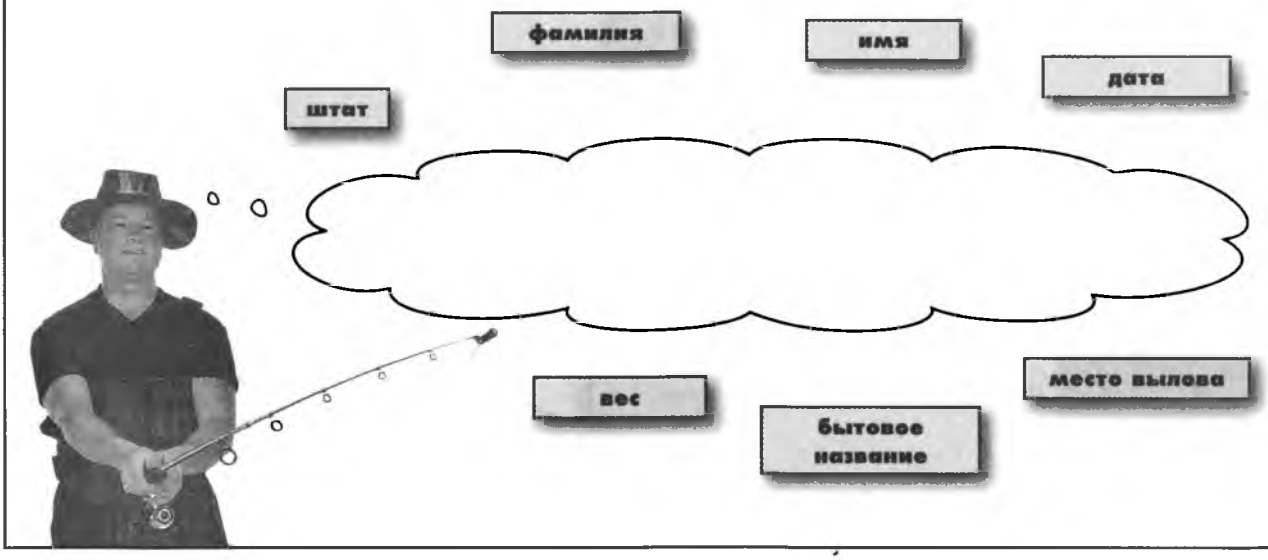
Как проще всего запросить данные из таблицы?

* Встречается мнение, что термин «РЕЛЯЦИОННЫЙ» относится к логическим связям между *таблицами*. Это неверно.

Упражнение

Сможете ли вы определить столбцы таблицы по тем словам, которыми ихтиолог Марк описывает выборку данных из таблицы? Запишите имена столбцов в прямоугольниках.

Теперь ваша очередь. Напишите аналогичную фразу для Джека, автора статей по рыбной ловле, который использует таблицу для получения подробной информации для своих статей. Затем проведите стрелки от каждого столбца к его упоминанию в описании.



Да, можно, но такой уровень детализации данных просто не нужен.

По крайней мере не в этом конкретном случае. Если бы Джек писал статьи о том, куда лучше отправиться на выходных, чтобы поймать большую рыбу, *тогда* он, возможно, указал бы название улицы и номер дома, чтобы читатели могли поискать жилье где-нибудь поблизости.

Но Джека интересует только место вылова и штат, и он добавил только эти столбцы, чтобы не увеличивать объем базы данных без необходимости. На этой стадии он решил, что его данные достаточно детализированы — то есть являются ***атомарными***.

Как вы думаете, что означает термин *атомарный* в контексте данных SQL?

Атомарные данные

Что такое «атом»? Маленький блок информации, который невозможно (или нежелательно) разделить на составные части меньшего размера. Это определение относится и к данным: АТОМАРНЫЕ данные были разделены на наименьшие компоненты, дальнейшее деление которых ***невозможно или нежелательно.***

Доставка за 30 минут, или Пицца бесплатно

Для примера возьмем курьера, доставляющего пиццу клиентам. Чтобы выполнить свою работу, ему достаточно знать улицу и номер дома в одном столбце. Для него эти данные являются атомарными: курьеру никогда не приходится искать номер дома отдельно от названия улицы.

Более того, разбиение адреса доставки на название улицы и номер дома только усложнит его работу, а клиентам придется дольше дожидаться своих заказов.

Для курьера адрес доставки, объединяющий улицу и номер дома в одном столбце, достаточно атомарен.

```
File Edit Window Help SimplePizzaFactory
+--------------+-----------------------------+
| order_number | address                     |
+--------------+-----------------------------+
| 246          | 59 N. Ajax Rapids           |
| 247          | 849 SQL Street              |
| 248          | 2348 E. PMP Plaza           |
| 249          | 1978 HTML Heights           |
| 250          | 24 S. Servlets Springs      |
| 251          | 807 Infinite Circle         |
| 252          | 32 Design Patterns Plaza    |
| 253          | 9208 S. Java Ranch          |
| 254          | 4653 W. EJB Estate          |
| 255          | 8678 OOA&D Orchard          |
+--------------+-----------------------------+
> SELECT address FROM pizza_deliveries WHERE order_num = 252;
+-----------------------------+
| address                     |
+-----------------------------+
| 32 Design Patterns Plaza    |
+-----------------------------+
1 row in set (0.04 sec)
```

С другой стороны

А теперь возьмем агента по торговле недвижимостью. Вполне возможно, что ему понадобится отдельный столбец с номером дома — допустим, чтобы он мог получить список всех предложений по заданной улице. Для него название улицы и номер дома являются атомарными данными.

Для агента по торговле недвижимостью ситуация выглядит иначе. Отделение номера дома от названия улицы позволит легко получить список домов, продаваемых на заданной улице.

File Edit Window Help IWantMyCommission

```
+---------------+-------------------------+---------------+----------+
| street_number | street_name             | property_type | price    |
+---------------+-------------------------+---------------+----------+
| 59            | N. Ajax Rapids          | condo         | 189000   |
| 849           | SQL Street              | apartment     | 109000   |
| 2348          | E. PMP Plaza            | house         | 355000   |
| 1978          | HTML Heights            | apartment     | 134000   |
| 24            | S. Servlets Springs     | house         | 355000   |
| 807           | Infinite Circle         | condo         | 143900   |
| 32            | Design Patterns Plaza   | house         | 465000   |
| 9208          | S. Java Ranch           | house         | 699000   |
| 4653          | SQL Street              | apartment     | 115000   |
| 8678          | OOA&D Orchard           | house         | 355000   |
+---------------+-------------------------+---------------+----------+
> SELECT price, property_type FROM real_estate WHERE street_name = 'SQL Street';
+-----------+---------------+
| price     | property_type |
+-----------+---------------+
| 109000.00 | apartment     |
| 115000.00 | apartment     |
+-----------+---------------+
2 rows in set (0.01 sec)
```

Атомарные данные и таблицы

Выбирая данные, которые будут храниться в ваших таблицах, задайте себе следующие вопросы.

1. Какой один объект описывает ваша таблица?

Что описывает ваша таблица: клоунов, коров, пончики, людей?

2. Как вы предполагаете использовать таблицу для получения информации о ее объекте?

Спроектируйте таблицу так, чтобы запросы были простыми!

3. Содержат ли столбцы таблицы атомарные данные, чтобы запросы были короткими и конкретными?

Часто Задаваемые Вопросы

В: Атомы совсем крошечные, верно? Значит ли это, что данные нужно разбить на *мельчайшие* фрагменты?

О: Нет. Атомарность данных подразумевает разбиение данных на наименьшие части, необходимые для создания эффективной таблицы, а не просто на самые мелкие части из всех возможных.

Не дробите данные сверх необходимости. Если лишние столбцы вам не нужны, не добавляйте их.

В: Как атомарность данных упростит мою работу?

О: Атомарность упрощает контроль за правильностью данных в таблице. Например, если в столбце хранятся номера домов, можно проследить за тем, чтобы в этом столбце хранились только числовые данные.

Кроме того, атомарность повышает эффективность запросов: запросы к атомарным данным быстрее пишутся и выполняются, что дает ощутимый эффект при хранении очень больших объемов данных.

Возьми в руку карандаш

Перед вами общепринятые правила определения атомарных данных. Для каждого правила приведите **два** гипотетических примера таблиц, нарушающих данное правило.

ПРАВИЛО 1. Столбец, содержащий атомарные данные, не может состоять из нескольких однотипных элементов.

Столбец interests таблицы Грега my_contacts нарушает это правило.

ПРАВИЛО 2. Таблица с атомарными данными не может содержать несколько однотипных столбцов.

Таблица easy_drinks нарушает это правило.

Теперь, когда вы знаете «официальные» правила атомарности и три этапа создания атомарных таблиц, взгляните на каждую таблицу, приведенную ранее в книге, и объясните, почему она является (или не является) атомарной.

Таблица Грега, с. 83 .....

Таблица с оценками пончиков, с. 112 .....

Таблица с клоунами, с. 155 .....

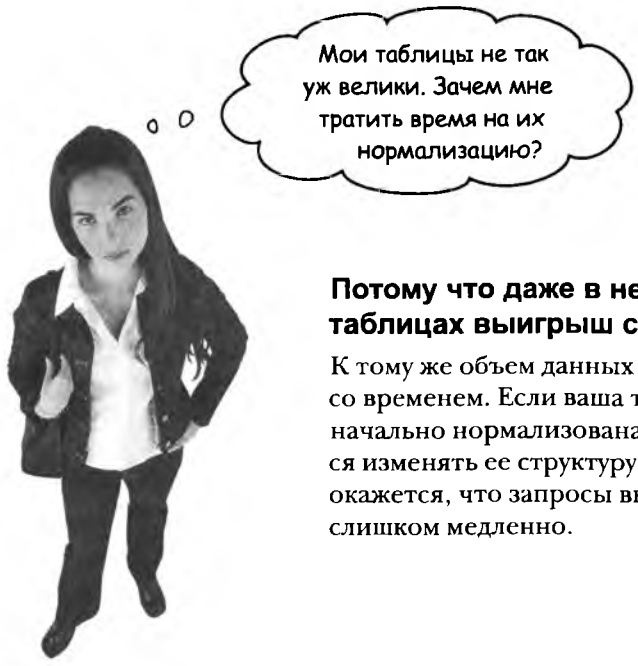
Таблица с описаниями напитков, с. 93 .....

Информация о рыбах, с. 194 .....

Преимущества нормализованных таблиц

1. **Нормализованные таблицы не содержат дубликатов данных, а это сокращает размер базы данных.**

2. **Уменьшение объема данных, по которым ведется поиск, ускоряет выполнение запросов.**



Потому что даже в небольших таблицах выигрыш суммируется.

К тому же объем данных увеличивается со временем. Если ваша таблица будет изначально нормализована, вам не придется изменять ее структуру позднее, когда окажется, что запросы выполняются слишком медленно.

Ненормализованные клоуны

Помните таблицу с информацией о клоунах? Сбор информации о клоунах неожиданно превратился в национальное увлечение, и старая таблица уже не справляется с потоком информации, потому что столбцы `appearance` и `activities` содержат *слишком* много данных. Для наших целей эта таблица не является атомарной.

Запросы с поиском к этим двум столбцам получаются очень сложными — столбцы содержат слишком много данных!

clown_info

name	last_seen	appearance	activities
Элси	Дом престарелых Черри Хилл	Ж, рыжие волосы, зеленый костюм, огромные ботинки	шарики, машинки
Пиклз	Вечеринка Джека Грина	М, оранжевые волосы, синий костюм, огромные ботинки	мим
Снаглз	Болмарт	Ж, желтая рубашка, синие штаны	рожок, зонтик
Мистер Хобо	Цирк BG	М, сигара, черные волосы, маленькая шляпа	скрипка
Кларабелл	Дом престарелых Бельмонт	Ж, розовые волосы, большой цветок, синее платье	кричалки, танцы
Скутер	Больница Окленд	М, синие волосы, красный костюм, большой нос	шарики
Зиппо	Торговый центр Милстоун	Ж, оранжевый костюм, штаны	танцы
Бэйб	Автошкола Эрла	Ж, розовый костюм с блестками	эквилибристика, машинки
Бонзо	Парк Диксон	М, женское платье в горошек	пение, танцы
Снифлз	Заведение Трэйси	М, зелено-фиолетовый костюм, длинный нос	разъезжает на машинке

Возьми в руку карандаш

Давайте сделаем таблицу более атомарной. Предположим, поиск должен вестись по столбцам `appearance` и `activities`, а также по столбцам `last_seen`. Запишите более правильную структуру столбцов.

Ответ на с. 229.

На полпути к 1НФ

Даже когда таблица содержит атомарные данные, пройдена только половина пути. Полоностью нормализованная таблица находится в ПЕРВОЙ НОРМАЛИЗОВАННОЙ ФОРМЕ, или сокращенно 1НФ.

Таблица, находящаяся в форме 1НФ, должна выполнять следующие два правила.

Мы уже знаем, как это делается.

Каждая запись должна содержать атомарные значения.

Чтобы таблица была полностью нормализована, каждой записи необходимо присвоить первичный ключ.

Каждая запись должна обладать уникальным идентификатором, который называется первичным ключом.

МОЗГОВОЙ ШТУРМ

Как вы думаете, какие столбцы хорошо подойдут на роль первичного ключа?

Правила первичных ключей

Столбец таблицы, который станет ее первичным ключом, назначается при создании таблицы. Через несколько страниц мы создадим таблицу и назначим первичный ключ, но сначала давайте повнимательнее разберемся с тем, какими свойствами должен обладать первичный ключ.

Первичный ключ — столбец таблицы, имеющий уникальное значение для каждой записи.

Первичный ключ используется для однозначной идентификации записей.

Это значит, что данные в столбце первичного ключа не могут повторяться. Для примера возьмем следующую таблицу; как вы думаете, какие из ее столбцов хорошо подойдут на роль первичного ключа?

SSN (номер социального страхования)	last_name (фамилия)	first_name (имя)	phone_number (телефон)

Каждому человеку назначается уникальный номер социального страхования; этот столбец может стать первичным ключом.

В этих трех столбцах с высокой вероятностью будут встречаться повторяющиеся значения — например, в базе данных могут быть записи нескольких людей по имени Джон, а несколько людей, живущих вместе, могут иметь одинаковые телефоны. Вероятно, это не лучшие кандидаты на роль первичного ключа.

Будьте осторожны!

Будьте осторожны при использовании номеров социального страхования в базах данных.

Количество краж личных данных только увеличивается, и люди неохотно сообщают свои коды социального страхования — и вполне обоснованно. Эти данные слишком важны, чтобы рисковать ими. Можете ли вы гарантировать, что ваша база данных защищена на 100%? Если нет — номера социального страхования могут быть похищены вместе с личными данными ваших клиентов.

Первичный ключ не может содержать NULL

Значение NULL не может быть уникальным, потому что в других записях этот столбец тоже может содержать NULL.

Значение первичного ключа должно задаваться при вставке записи

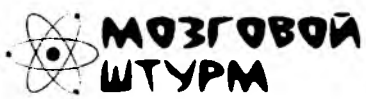
При вставке в таблицу записи без указания значения первичного ключа возникает риск создания записи с первичным ключом NULL и появления дубликатов, а это нарушает требования первой нормальной формы.

Первичный ключ должен быть компактным

Первичный ключ должен содержать только ту информацию, которая обеспечивает его уникальность, и ничего более.

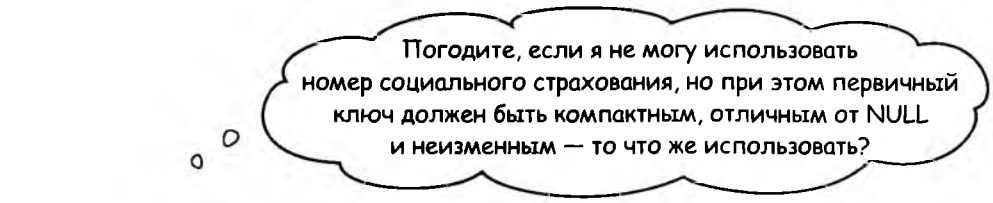
Значения первичного ключа должны оставаться неизменными

Если бы первичный ключ можно было изменять, то ему можно было бы случайно присвоить уже используемое значение. Помните, что первичный ключ должен быть уникальным.



Сможете ли вы предложить хороший первичный ключ с учетом всех этих правил?

Еще раз просмотрите таблицы, встречавшиеся нам в книге. Есть ли в какой-либо из них столбец, содержащий уникальные значения?



Лучшим первичным ключом может быть *новый первичный ключ.*

В том, что касается первичных ключей, лучшим решением часто оказывается создание столбца, содержащего уникальный номер. Представьте таблицу, которая содержит все прежние данные, к которым добавляется новый числовой столбец. В следующем примере он будет называться `ID` (идентификатор).

Если бы не столбец `ID`, две записи Джона Брауна были бы одинаковыми, но в данном случае речь идет о двух разных людях. Столбец `ID` обеспечивает уникальность этих записей. Таблица находится в первой нормальной форме.

id	last_name	first_name	nick_name
1	Браун	Джон	Джон
2	Элсуорт	Ким	Ким
3	Браун	Джон	Джон
4	Петрильо	Мария	Мария
5	Франкен	Эсме	Эм

Запись Джона Брауна.

Тоже запись Джона Брауна, но столбец ID показывает, что эта уникальная запись относится к другому Джону Брауну.

Для любознательных

В мире SQL идут ожесточенные споры по поводу использования синтетических (то есть искусственно созданных, как столбец `ID` в этом примере) и естественных ключей — данных, уже хранящихся в таблице (номер машины, номер социального страхования и т. д.). Мы не будем становиться на ту или иную сторону; в главе 7 первичные ключи будут рассмотрены более подробно.

Часто Задаваемые Вопросы

В: Вы упоминаете о «первой» нормальной форме. Значит, есть и вторая? И третья?

О: Да, вторая и третья нормальные формы действительно существуют; они определяются более жесткими правилами. Вторая и третья нормальные формы рассматриваются в главе 7.

В: Мы изменили свои таблицы, чтобы в них хранились атомарные значения. Какая-нибудь из этих таблиц находится в 1НФ?

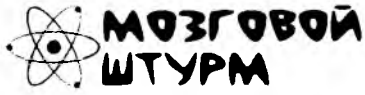
О: Нет. До настоящего момента ни одна из созданных нами таблиц не имела первичного ключа с уникальными значениями.

В: Столбец `comments` в таблице с описаниями пончиков мне не кажется атомарным. Другими словами, я не вижу, как удобно провести поиск по этому столбцу.

О: Совершенно верно. Поле не особенно атомарно, но структура нашей таблицы этого и не требует. Если бы мы захотели ограничить комментарии заранее определенным набором слов, то поле могло бы стать атомарным. С другой стороны, тогда поле не содержало бы искренние комментарии в произвольной форме.

Как прийти в НОРМУ

Пришло время отступить на шаг и нормализовать наши таблицы. Для этого необходимо сделать данные атомарными и назначить первичные ключи. Создание первичного ключа — один из стандартных этапов написания кода команды `CREATE TABLE`.



А вы помните, как добавить столбец в существующую таблицу?

Исправление таблицы Грега

После всего сказанного становится ясно, что необходимо сделать для исправления таблицы Грега.

Исправление таблицы Грега, шаг 1. Выполнить выборку всех данных командой SELECT и как-то сохранить их.

Исправление таблицы Грега, шаг 2. Создать новую нормализованную таблицу.

Исправление таблицы Грега, шаг 3. Вставить все старые данные в новую таблицу, изменяя каждую запись в соответствии с новой структурой таблицы.

Теперь старую таблицу можно удалить.

Конечно, таблица Грега не совершенна.

Она не атомарна и в ней нет первичного ключа. Но Грегу повезло: ему ***не придется*** пользоваться старой таблицей и не придется удалять данные.

Чтобы назначить в таблице Грега первичный ключ и сделать столбцы более атомарными, достаточно всего одной команды. Но сначала небольшое отступление...

Старая команда CREATE TABLE

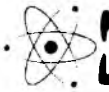
Таблице Грега нужен первичный ключ. После всех разговоров об атомарности данных Грег понимает, что он может принять меры для того, чтобы сделать столбцы своей таблицы более атомарными. Но прежде чем разбираться с тем, как исправить существующую таблицу, давайте вспомним, как она создавалась!

Вот как выглядела команда создания таблицы из главы 1.

```
CREATE TABLE my_contacts
(
  last_name VARCHAR(30),
  first_name VARCHAR(20),
  email VARCHAR(50),
  gender CHAR(1),
  birthday DATE,
  profession VARCHAR(50),
  location VARCHAR(50),
  status VARCHAR(20),
  interests VARCHAR(100),
  seeking VARCHAR(100)
);
```

Нет подходящего кандидата на роль первичного ключа.

Нельзя ли сделать эти столбцы более атомарными при создании таблицы?



МОЗГОВОЙ ШТУРМ

А если старая команда `CREATE TABLE` нигде не была записана? Как получить доступ к коду создания таблицы?

Сначала покажи ~~деньги~~ таблицу

Может, для просмотра кода создания таблицы воспользоваться командой `DESCRIBE my_contacts`? Результат ее выполнения будет выглядеть примерно так:

```
File Edit Window Help GregsListAgain
+-------------+---------------+------+-----+---------+----------------+
| Column      | Type          | Null | Key | Default | Extra          |
+-------------+---------------+------+-----+---------+----------------+
| last_name   | varchar(30)   | YES  |     | NULL    |                |
| first_name  | varchar(20)   | YES  |     | NULL    |                |
| email       | varchar(50)   | YES  |     | NULL    |                |
| gender      | char(1)       | YES  |     | NULL    |                |
| birthday    | date          | YES  |     | NULL    |                |
| profession  | varchar(50)   | YES  |     | NULL    |                |
| location    | varchar(50)   | YES  |     | NULL    |                |
| status      | varchar(20)   | YES  |     | NULL    |                |
| interests   | varchar(100)  | YES  |     | NULL    |                |
| seeking     | varchar(100)  | YES  |     | NULL    |                |
+-------------+---------------+------+-----+---------+----------------+
```

Но нас интересует код **CREATE**, а не описания полей таблицы. И нам хотелось бы узнать, как должна выглядеть исходная команда, не вводя ее заново.

Команда `SHOW CREATE_TABLE` возвращает команду `CREATE TABLE`, которая была использована для создания таблицы (до занесения в таблицу первых данных). Попробуйте ввести следующую команду:

```
SHOW CREATE TABLE my_contacts;
```

Команда для экономии времени

Взгляните на код, который использовался для создания таблицы на с. 217, и приведенный ниже результат выполнения команды `SHOW CREATE TABLE my_contacts`. Эти фрагменты не идентичны, но если вставить этот код в команду `CREATE TABLE`, результат будет тем же. Удалять обратные апострофы или параметры данных не нужно, но если вы это сделаете, команда получится более компактной.

Имена столбцов и таблицы заключены в обратные апострофы. Эти символы присутствуют в результатах команды SHOW CREATE TABLE.

```
CREATE TABLE `my_contacts`
(
    `last_name` varchar(30) default NULL,
    `first_Name` varchar(20) default NULL,
    `email` varchar(50) default NULL,
    `gender` char(1) default NULL,
    `birthday` date default NULL,
    `profession` varchar(50) default NULL,
    `location` varchar(50) default NULL,
    `status` varchar(20) default NULL,
    `interests` varchar(100) default NULL,
    `seeking` varchar(100) default NULL,
) ENGINE=MyISAM DEFAULT CHARSET=cp1251
```

SQL считает, что столбцы по умолчанию инициализируются значением NULL (если явно не задано другое значение).

При создании таблицы желательно указывать, может ли столбец содержать NULL.

Не обращайте внимания на текст после закрывающей круглой скобки. Он описывает механизм хранения данных и используемую кодировку символов. Пока нас устроят значения по умолчанию.

Если исходная таблица не была удалена, то этой таблице придется присвоить новое имя.

Если скопировать и выполнить этот код, он создаст таблицу.

Команда CREATE TABLE с назначением первичного ключа

Перед вами код, полученный при выполнении команды `SHOW CREATE TABLE`. Мы удалили из него обратные апострофы и последнюю строку. В начало списка столбцов был добавлен столбец `contact_id` с условием `NOT NULL`, а в конце списка появилось условие `PRIMARY KEY`, в котором новый столбец `contact_id` назначается первичным ключом.

Помните, что столбец первичного ключа не может содержать NULL! Присутствие NULL в столбце первичного ключа не позволит однозначно идентифицировать каждую запись в таблице.

```
CREATE TABLE my_contacts
(
  contact_id INT NOT NULL,
  last_name varchar(30) default NULL,
  first_name varchar(20) default NULL,
  email varchar(50) default NULL,
  gender char(1) default NULL,
  birthday date default NULL,
  profession varchar(50) default NULL,
  location varchar(50) default NULL,
  status varchar(20) default NULL,
  interests varchar(100) default NULL,
  seeking varchar(100) default NULL,
  PRIMARY KEY (contact_id)
)
```

Мы создали новый столбец contact_id, который станет первичным ключом таблицы. Хранящиеся в нем целые числа уникальны для каждой записи, а таблица становится атомарной.

Здесь назначается первичный ключ таблицы. Синтаксис прост: за ключевыми словами PRIMARY KEY в круглых скобках указывается имя столбца, который будет первичным ключом — в нашем примере это новый столбец contact_id.

Часто Задаваемые Вопросы

В: Вы говорите, что первичный ключ не может содержать NULL. Что еще предотвращает появление в нем дубликатов?

О: Прежде всего вы сами. При вставке значений в таблицу столбцу `contact_id` присваиваются уникальные значения. Например, в первой команде `INSERT` столбцу `contact_id` присваивается значение `1`, во второй — значение `2` и т. д.

В: Присваивать новое значение столбцу PRIMARY KEY при каждой вставке новой записи весьма хлопотно. Нет ли более простого способа?

О: Есть два таких способа. Первый — использование в качестве первичного ключа заведомо уникального столбца таблицы. Мы уже упоминали о том, что этот способ может создать проблемы (как, например, при использовании номеров социального страхования).

Второй, более простой способ заключается в создании нового столбца с уникальными идентификаторами — как, например, `contact_id` на предыдущей странице. Вы можете приказать своей РСУБД автоматически генерировать его значения при помощи специальных ключевых слов (подробности на следующей странице).

В: Для чего еще можно использовать SHOW, кроме вывода команды CREATE?

О: Команда `SHOW` может использоваться для вывода информации о столбцах таблицы:

```
SHOW COLUMNS FROM tablename;
```

Команда выводит описания всех столбцов таблицы с типами данных, а также другими сведениями, относящимся к конкретным столбцам.

```
SHOW CREATE DATABASE databasename;
```

По аналогии с командой `SHOW CREATE <таблица>`, эта команда выводит код команды создания базы данных.

```
SHOW INDEX FROM tablename;
```

Команда выводит информацию об индексируемых столбцах и типах индексов. До настоящего момента из индексов нам встречались только первичные ключи, но скоро вы лучше поймете смысл этой команды.

И еще одна ОЧЕНЬ полезная команда:

```
SHOW WARNINGS;
```

Если на консоли выводится сообщение о том, что выполнение команды SQL привело к выдаче предупреждений, то для просмотра предупреждений используется команда `SHOW WARNINGS`.

Существуют и другие разновидности команды `SHOW`. Мы рассмотрели лишь те, которые имеют прямое отношение к интересующим нас темам.

В: Для чего нужны обратные апострофы в результатах SHOW CREATE TABLE? Вы уверены, что без них можно обойтись?

О: РСУБД в некоторых ситуациях не может определить, что имя столбца действительно является именем столбца. Например, если имена столбцов будут заключаться в обратные апострофы, вы сможете использовать в качестве имен зарезервированные ключевые слова SQL (хотя это крайне неудачная мысль).

Допустим, по каким-то непостижимым причинам вы хотите включить в таблицу столбец с именем `select`. Такое объявление столбца недопустимо:

```
select varchar(50)
```

А такое объявление сработает:

```
`select` varchar(50)
```

В: А почему ключевые слова нельзя использовать в именах столбцов?

О: Можно, но нежелательно. Только представьте, какими запутанными станут ваши запросы и сколько хлопот будет с вводом обратных апострофов, когда можно обойтись без них. Кроме того, `select` — неудачное имя столбца: оно ничего не сообщает о данных, которые в нем хранятся.

1, 2, 3 и так далее

Если снабдить столбец `contact_id` ключевым словом `AUTO_INCREMENT`, то РСУБД будет автоматически заполнять его значениями: 1 для записи 1, 2 для записи 2 и т. д.

```
CREATE TABLE my_contacts
(
    contact_id INT NOT NULL AUTO_INCREMENT,
    last_name varchar(30) default NULL,
    first_name varchar(20) default NULL,
    email varchar(50) default NULL,
    gender char(1) default NULL,
    birthday date default NULL,
    profession varchar(50) default NULL,
    location varchar(50) default NULL,
    status varchar(20) default NULL,
    interests varchar(100) default NULL,
    seeking varchar(100) default NULL,
    PRIMARY KEY (contact_id)
)
```

Вот оно: в большинстве реализаций SQL просто добавьте ключевое слово AUTO_INCREMENT (Пользователи MS SQL указывают ключевое слово INDEX с начальным значением и приращением. За конкретной информацией обращайтесь к справочному руководству по MS SQL).

У первой записи в этом столбце сохраняется значение 1. Затем значение столбца автоматически увеличивается на 1 при каждой вставке новой записи.

Пока все достаточно просто. Но как должна выглядеть команда INSERT, если этот столбец заполняется автоматически? Могу ли я случайно присвоить ему другое значение?

Как вы думаете, что произойдет?

А еще лучше – попробуйте и посмотрите сами.

Упражнение

1. Напишите команду `CREATE TABLE` для создания приведенной ниже таблицы, в которой хранятся имена и фамилии. Таблица должна содержать столбец первичного ключа с ключевым `AUTO_INCREMENT` и два атомарных столбца.

2. Откройте терминал SQL или графический интерфейс, выполните команду `CREATE TABLE`.

3. Попробуйте выполнить каждую из приведенных ниже команд `INSERT`. Обведите кружком команды, которые были успешно выполнены.

```
INSERT INTO your_table (id, first_name, last_name)
VALUES (NULL, 'Марсия', 'Брэди');

INSERT INTO your_table (id, first_name, last_name)
VALUES (1, 'Джен', 'Брэди');

INSERT INTO your_table
VALUES ('', 'Бобби', 'Брэди');

INSERT INTO your_table (first_name, last_name)
VALUES ('Синди', 'Брэди');

INSERT INTO your_table (id, first_name, last_name)
VALUES (99, 'Питер', 'Брэди');
```

4. Все ли команды были выполнены успешно? Напишите, как будет выглядеть содержимое таблицы после выполнения команд `INSERT`.

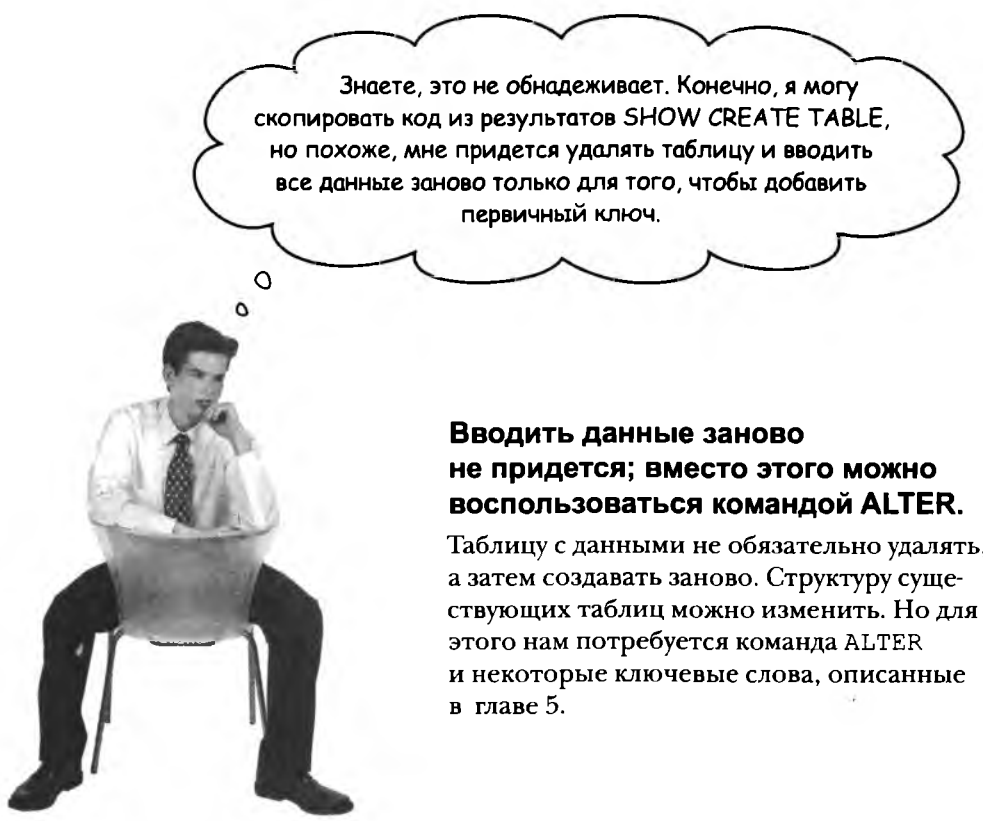
your_table

id	first_name	last_name

Часто Задаваемые Вопросы

В: Почему первый запрос (с NULL в столбце id) вставляет запись, хотя для id установлено ограничение NOT NULL?

О: Хотя на первый взгляд команда выполняться не должна, с AUTO_INCREMENT значение NULL просто игнорируется. С другой стороны, без AUTO_INCREMENT вы получите сообщение об ошибке, а запись вставлена не будет. Убедитесь в этом сами.



Вводить данные заново не придется; вместо этого можно воспользоваться командой ALTER.

Таблицу с данными не обязательно удалять, а затем создавать заново. Структуру существующих таблиц можно изменить. Но для этого нам потребуется команда ALTER и некоторые ключевые слова, описанные в главе 5.

Добавление первичного ключа в существующую таблицу

Перед вами код добавления первичного ключа `AUTO_INCREMENT` в таблицу `my_contacts`. (Команда получается довольно длинной, так что книгу придется развернуть.)

```
ALTER TABLE my_contacts
ADD COLUMN contact_id INT NOT NULL AUTO_INCREMENT FIRST,
ADD PRIMARY KEY (contact_id);
```

Новая команда SQL: ALTER.

Код добавления нового столбца в таблицу. Выглядит знакомо, не правда ли?

FIRST приказывает РСУБД поставить новый столбец на первое место в списке. Строго говоря, это не обязательно, но нахождение первичного ключа в начале списка считается «хорошим стилем».

Ключевые слова ADD COLUMN сообщают, что в таблицу добавляется новый столбец с именем contact_id.

Вероятно, вы узнали строку, в которой назначается первичный ключ.

МОЗГОВОЙ ШТУРМ

Как вы думаете, создаст ли эта команда значения нового столбца `contact_id` для записей, уже находящихся в таблице, или же они будут создаваться только для вновь вставляемых записей? Как это проверить?

ALTER TABLE и добавление первичного ключа

Проверьте, как работает этот код. Откройте терминал SQL, выполните команду `USE` для базы данных `gregs_list` и введите следующую команду:

```
File Edit Window Help Alterations
> ALTER TABLE my_contacts
    -> ADD COLUMN contact_id INT NOT NULL AUTO_INCREMENT FIRST,
    -> ADD PRIMARY KEY (contact_id);

Query OK, 50 rows affected (0.04 sec)
Records: 50  Duplicates: 0  Warnings: 0
```

Сообщает, что столбец был добавлен в 50 записях, уже хранящихся в нашей таблице. У вас их будет меньше.

Чтобы увидеть, что произошло с таблицей, выполните команду **`SELECT * from my_contacts;`**

Столбец contact_id включен в таблицу первым, до всех остальных столбцов.

```
File Edit Window Help Alterations
+------------+------------+------------+-----------------------
| contact_id | last_name  | first_name | email
+------------+------------+------------+-----------------------
|          1 | Андерсон   | Джиллиан   | jill_anderson@yahoo.c
|          2 | Иоффе      | Кевин      | kj@simuduck.com
|          3 | Ньюсам     | Аманда     | aman2luv@yahoo.com
|          4 | Гарсиа     | Эд         | ed99@mysoftware.com
|          5 | Раундтри   | Джо-Энн    | jojo@yahoo.com
|          6 | Бриггс     | Крис       | cbriggs@mail.com
```

Так как мы использовали AUTO_INCREMENT, столбец автоматически заполняется значениями при обновлении записей таблицы.

При следующей вставке новой записи столбцу contact_id будет присвоено значение, на 1 больше максимального значения contact_id в таблице. Если у последней записи столбец contact_id содержит значение 23, то у следующей записи он будет равен 24.

Напомним, что это еще не конец таблицы; у Грега много знакомых.

Получит ли Грег свой столбец с номером телефона? Об этом вы узнаете в главе 5.

ГЛАВА 4

Новые инструменты

Вы взяли на вооружение материал главы 4. Только посмотрите, сколько у вас появилось новых инструментов! Полный список инструментов приведен в приложении III.