SQL для начинающих. Часть 3

*Представляю Вашему вниманию вольный перевод статьи*[*SQL for Beginners Part 3*](http://net.tutsplus.com/tutorials/databases/sql-for-beginners-part-3/)

Сегодня мы продолжаем наше путешествие в мире SQL и реляционных систем управления базами данных. В этой части мы научимся работать с несколькими таблицами связанными между собой. Сначала мы познакомимся с базовыми концепциями, а потом начнем работать с запросами JOIN в SQL.

**Предыдущие статьи**

* [SQL для начинающих. Часть 1](http://jtest.ru/bazyi-dannyix/sql-dlya-nachinayushhix.html)
* [SQL для начинающих. Часть 2](http://jtest.ru/bazyi-dannyix/sql-dlya-nachinayushhix-chast-2.html)

**Вступление**

При проектировании базы данных, здравый смысл подсказывает нам, что мы должны использовать различные таблицы для разных данных. Пример: клиенты, заказы, записи, сообщения и т.д. Так же мы должны иметь взаимосвязи между этими таблицами. Например, клиент имеет заказы, а у заказа есть позиции (товары). Эти взаимосвязи должны быть отражены в базе данных. А также, когда мы получаем данные с помощью SQL, мы должны использовать определенные типы запросов JOIN, чтобы получить нужный результат.

Вот несколько типов отношений в базе данных. Сегодня мы рассмотрим следующие:

* Отношения один к одному
* Один ко многим и многие к одному
* Многие ко многим
* Связь с самим собой

Когда данные выбираются из нескольких связанных таблиц, мы будем использовать запрос *JOIN*. Есть несколько типов присоединения, мы познакомимся с этими:

* Cross Joins (Перекрестное соединение)
* Natural Joins (Естественное соединений)
* Inner Joins (Внутреннее соединений)
* Left (Outer) Joins (Левое (внешнее) соединение)
* Right (Outer) Joins (Правое (внешнее) соединение)

Также мы изучим предложения *ON* и *USING*.

**Связь один к одному**

Допустим есть таблица покупателей (customers):



Мы можем расположить информацию о адресе покупателя в другой таблице:



Теперь у нас есть связь между таблицами покупателей (Customers) и адресами (Addresses). Если каждый адрес может принадлежать только одному покупателю, то такая связь называется "Один к одному". Имейте ввиду, что такой тип отношений не очень распространен. Наша первоначальная таблица, в которой информация о покупателе и его адресе хранилась вместе, в большинстве случаев работает нормально.

Обратите внимание, что теперь поле с названием "address\_id", в таблице покупателей, ссылается на соответствующую запись в таблице адресов. Оно называется внешним ключом (Foreign Key) и используется во всех видах связей в базе. Мы рассмотрим этот вопрос позже в этой статье.

Вот так можно отобразить отношения между покупателями и адресами:



Обратите внимание, что существование данных отношений не обязательно, например, может существовать запись о покупателе без связанной записи о его адресе.

**Связь один ко многим и многие к одному**

Этот тип отношений наиболее часто встречающийся. Рассмотрим такой сайт интернет магазина:

* У покупателей может быть несколько заказов.
* Заказ может содержать несколько товаров.
* Товары могут иметь описание на нескольких языках.

В этих случаях нам потребуется создать связь "Один ко многим". Пример:



Каждый покупатель может иметь 0 или более заказов. Но каждый заказ может принадлежать только одному покупателю.



**Связь многие ко многим**

В некоторых случаях требуется многочисленные связи по обе стороны отношений. Например, каждый заказ может содержать множество товаров. И каждый товар может присутствовать во многих заказах.

Для такой связи нам потребуется создать дополнительную таблицу:



Назначение таблицы "Items\_Orders" только одно - создать связь "Многие ко многим" между товарами и заказами.

Так можно представить этот тип отношений:



Если добавить записи items\_orders к диаграмме, то она будет выглядеть так:



**Связь с собой**

Такой тип используется когда у таблицы должны быть связь с собой. Допустим у Вас есть реферальная программа. Покупатели могут ссылаться на других покупателей на вашем сайте интернет магазина. Таблица может выглядеть так:



Покупатели 102 и 103 ссылаются на покупателя 101.

Этот тип похож на связь "Один ко многим", поскольку один покупатель может ссылаться на несколько покупателей. Это можно представить как древовидную структуру:



Один покупатель может ссылаться на одного покупателя, на нескольких покупателей, или вообще не ссылаться ни на одного.

Если Вы хотите создать связь внутри таблицы "многие ко многим", то потребуется создать дополнительную таблицу, такую же как и в предыдущей части.

**Внешние ключи**

Пока что мы говорили только о базовых вещах. Пришло время применить полученные знания на практике, используя SQL. В данной часть нам нужно понять, что из себя представляют внешние ключи.

В отношениях, обсуждаемых выше, у нас всегда было поле вида "\*\*\*\*\_id", которое ссылалось столбец в другой таблице. В нашем примере столбец customer\_id, в таблице Orders, является внешним ключом:



В таких базах как MySQL есть два способа создания внешних ключей:

**Задать внешний ключ явно**

Создадим простую таблицу с покупателями:

[?](http://jtest.ru/)

|  |  |
| --- | --- |
| 1234 | CREATE TABLE customers (    customer\_id INT AUTO\_INCREMENT PRIMARY KEY,    customer\_name VARCHAR(100)); |

Теперь создадим таблицу заказов, которая будет содержать вторичный ключ:

[?](http://jtest.ru/)

|  |  |
| --- | --- |
| 123456 | CREATE TABLE orders (    order\_id INT AUTO\_INCREMENT PRIMARY KEY,    customer\_id INT,    amount DOUBLE,    FOREIGN KEY (customer\_id) REFERENCES customers(customer\_id)); |

Оба столбца (customers.customer\_id и orders.customer\_id) должны быть одного типа. Если у первого тип INT, то второй не должен быть типа BIGINT, например.

Пожалуйста, помните, что в MySQL полностью поддерживает внешние ключи только подсистема InnoDB. Другие подсистемы хранения данных позволяют определять внешние ключи без каких либо ошибок. Столбцы с внешними ключами индексируются автоматически, если явно не задать другой индекс.

**Без явного объявления**

Некоторые таблицы заказов могут быть созданы без явного определения внешнего ключа:

[?](http://jtest.ru/)

|  |  |
| --- | --- |
| 123456 | CREATE TABLE orders (    order\_id INT AUTO\_INCREMENT PRIMARY KEY,    customer\_id INT,    amount DOUBLE,    INDEX (customer\_id)); |

Когда данные получают запросом *JOIN*, Вы можете использовать столбец как внешний ключ, хотя база данных не знает о этих связях.

[?](http://jtest.ru/)

|  |  |
| --- | --- |
| 12 | SELECT \* FROM ordersJOIN customers USING(customer\_id) |

Мы подошли к изучению запросов *JOIN*, которые обсудим далее в статье.

**Отображение связей**

В данный момент, моей любимой программой для проектирования баз данных и отображения связей является [MySQL Workbench](http://wb.mysql.com/).



После того как Вы спроектировали базу данных, ее можно экспортировать в SQL и выполнить на сервере. Это очень удобно при создании больших и сложных баз данных.



**Запросы JOIN**

Чтобы получить связанные данные из базы данных следует использовать запросы *JOIN*.

Прежде чем мы начнем, давайте создадим для работы тестовые таблицы и данные.

[?](http://jtest.ru/)

|  |  |
| --- | --- |
| 1234567891011121314151617181920212223 | CREATE TABLE customers (    customer\_id INT AUTO\_INCREMENT PRIMARY KEY,    customer\_name VARCHAR(100)); CREATE TABLE orders (    order\_id INT AUTO\_INCREMENT PRIMARY KEY,    customer\_id INT,    amount DOUBLE,    FOREIGN KEY (customer\_id) REFERENCES customers(customer\_id)); INSERT INTO `customers` (`customer\_id`, `customer\_name`) VALUES(1, 'Adam'),(2, 'Andy'),(3, 'Joe'),(4, 'Sandy'); INSERT INTO `orders` (`order\_id`, `customer\_id`, `amount`) VALUES(1, 1, 19.99),(2, 1, 35.15),(3, 3, 17.56),(4, 4, 12.34); |

У нас есть 4 покупателя. У одного из них два заказа, у двоих по одному заказу, и у одного вообще нет заказов. Теперь давайте посмотрим какие виды запросов *JOIN* мы можем выполнять с этими таблицами.

**Cross Join (Перекрестное объединение)**

Это вид *JOIN* запроса по-умолчанию, если не определено условие.



Результатом будет, так называемое, "Декартово объединение" таблиц. Это означает, что каждая строка из первой таблицы сопоставляется с каждой строкой второй таблицы. Т.к. в каждой таблице по 4 строки, мы получили в результате 16 строк.

Ключевое слово *JOIN* можно заменить на запятую, в этом случае.



Конечно такой результат почти бесполезен. Давайте взглянем на другие виды объединений.

**Natural Join (Естественное объединение)**

При таком виде запроса *JOIN* таблицы должны иметь совпадающие, по имени, столбцы. В нашем случае в обеих таблицах должен присутствовать столбец customer\_id. MySQL объединит записи только в случае совпадения значений в этих столбцах.



Как Вы можете видеть, в этот раз столбец customer\_id отображаются только один раз, потому что движок базы рассматривает этот столбец как общий. Мы видим два заказа Adam'а, и другие два заказа Joe и Sandy. Наконец мы получили некоторую полезную информацию.

**Inner Join (Внутреннее объединение)**

Если условие объединения не указан, то выполняется внутреннее объединение. В этом случае хорошей идеей будет наличие совпадений по полю customer\_id в обеих таблицах. Результат должен быть аналогичен естественному объединению.



Результат почти такой же. Столбец customer\_id повторяется два раза, по разу для каждой таблицы. Объясняется это тем, что мы попросили базу сравнить значение по двум столбцам. При этом не знаю, что возвращают одну и туже информацию.

Добавим побольше условий к запросу.



На этот раз возвращается только те заказы, сумма которых превышает $15.

**Предложение ON**

Прежде чем перейти к другим видам объединяющих запросов, нам нужно рассмотреть предложение *ON*. Оно служит для вставки условий *JOIN* в отдельные предложения.



Теперь мы можем различать условия, относящиеся к *JOIN* и условия в части *WHERE*. Но еще есть небольшая разница в функционировании. Мы увидим это, когда перейдем к примерам с *LEFT JOIN*.

**Предложение USING**

Предложение *USING* немного похоже на конструкцию *ON*. Если столбцы в таблицах называется одинаково, можно указать их здесь.



На самом деле это очень похоже на *NATURAL JOIN*, т.е. объединяющий столбец (customer\_id) не повторяется дважды.

**Left (Outer) Join (Левое внешнее соединение)**

LEFT JOIN это вид внешнего соединения. В следующем запросе, если не найдены совпадения во второй таблице, записи из первой таблице все равно отобразятся.



Хотя у Andy и нет заказов, эта запись все равно отображается. Значение из второй таблицы равно NULL.

Это полезно, когда нужно найти записи, у которых нет связей. Например, мы можем найти всех покупателей, которые ничего не заказывали.



Все что мы сделали - нашли все значения NULL для order\_id.

Отметим, что ключевое слово *OUTER* не обязательно. Вы можете использовать просто *LEFT JOIN*вместо *LEFT OUTER JOIN*.

**Условия**

Теперь давайте посмотрим на запросы с условиями.



Так, что случилось с Andy и Sandy? *LEFT JOIN* подразумевает, что мы должны получить покупателей, у которых нет заказов. Проблема в том, что условие *WHERE* скрывает эти результаты. Чтобы получить их, мы можем попытаться включить условие с NULL.



Появился Andy, но нет Sandy. Выглядит неправильно. Для того чтобы получить то, что мы хотим, нужно использовать предложение *ON*.



Теперь мы получили всех, и все заказы более $15. Как я говорил ранее, предложение *ON*иногда работает не так как *WHERE*. В таких внешних объединениях как это, столбцы включаются всегда, даже если нет совпадений в условии предложения *ON*.

**Right (Outer) Join (Правое внешнее соединение)**

Объединение *RIGHT OUTER JOIN* работает также, только порядок таблиц меняется на обратный.



На этот раз мы не получили результатов с NULL, потому что каждый заказ имеет сопоставление с записью покупателя. Мы можем поменять порядок таблиц и получим тот же результат, что и с *LEFT OUTER JOIN*.

Теперь у нас появились значения NULL, потому что таблица покупателей с правой стороны от объединения.

**Заключение**

Спасибо за чтение статьи. Надеюсь Вам понравилось!