# SQL за 20 минут

От

**[ardernord](https://proglib.io/p/author/ardernord/)**

 -

23.02.2017

[0](https://proglib.io/p/sql-for-20-minutes/" \l "respond)

624

* [56](http://vk.com/share.php?url=https://proglib.io/p/sql-for-20-minutes/&title=%D0%98%D0%B7%D1%83%D1%87%D0%B0%D0%B5%D0%BC SQL %D0%B7%D0%B0 20 %D0%BC%D0%B8%D0%BD%D1%83%D1%82 - %D0%91%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0 %D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%81%D1%82%D0%B0&description=&image=&utm_source=share2" \o "ВКонтакте" \t "/home/pv/Documents\\x/_blank)
* [11](https://www.facebook.com/sharer.php?src=sp&u=https://proglib.io/p/sql-for-20-minutes/&utm_source=share2" \o "Facebook" \t "/home/pv/Documents\\x/_blank)

Предлагаем вашему вниманию статью с кричащим названием «SQL за 20 минут». Конечно, весь SQL за 20 минут вы не освоите, но хороший старт получите.

Каждый уважающий себя веб-разработчик должет знать [SQL](https://ru.wikipedia.org/wiki/SQL" \t "/home/pv/Documents\\x/_blank). Хоть он и существует аж с 70-х годов прошлого века, он до сих пор очень широко используется, и без него будет сложно создать нечто серьёзное. Большинство full-stack фреймворков умеют работать с SQL. В их числе: [ActiveRecord](http://guides.rubyonrails.org/active_record_basics.html" \t "/home/pv/Documents\\x/_blank), [Doctrine](http://www.doctrine-project.org/" \t "/home/pv/Documents\\x/_blank), [Hibernate](http://hibernate.org/orm/" \t "/home/pv/Documents\\x/_blank) и многие другие. Несмотря на это, иногда приходится «замарать руки» и пуститься в настоящий SQL.

Именно поэтому мы подготовили короткое вступление, в котором мы пройдёмся по фундаментальным вещам в SQL. Мы настоятельно рекомендуем вам попробовать все приведённые ниже примеры самостоятельно, ведь, как известно, теория — ничто без практики.

Что ж, приступим!

# Создаём таблицу

Для того, чтобы создать таблицу в SQL, используется выражение CREATE TABLE. Он принимает в качестве параметров все колонки, которые мы хотим внести, а также их типы данных.

Давайте создадим табличку с названием «Months», в которой будет три колонки:

* id — иными словами, порядковый номер месяца (целочисленный тип или int)
* name — название месяца (строка или varchar(10) (10 символов — максимальная длина строки))
* days — число дней в конкретном месяце (целочисленный тип или int)

Код будет выглядеть вот так:

|  |  |
| --- | --- |
| 1 | CREATE TABLE months (id int, name varchar(10), days int); |

Также, когда создаются таблицы, принято добавлять так называемый primary key. Это колонка, значения в которой уникальны. Чаще всего primary key колонкой является id, но в нашем случае это может быть и name, так как имена всех месяцев уникальны. Для более подробной информации предлагаем перейти по этой [ссылке](https://www.w3schools.com/sql/sql_primarykey.asp" \t "/home/pv/Documents\\x/_blank).

# Ввод данных

Теперь давайте добавим пару месяцев в нашу табличку. Сделать это можно с помощью команды INSERT. Есть два разных способа использовать INSERT:

Первый способ не подразумевает указания названий колонок, а лишь принимает значения в том порядке, в котором они указаны в таблице.

|  |  |
| --- | --- |
| 1 | INSERT INTO months VALUES (1,'January',31); |

Первый способ короче второго, однако если в будущем мы захотим добавить дополнительные колонки, все предыдущие запросы работать не будут. Для решения данной проблемы следует использовать второй способ. Его суть в том, что перед вводом данных мы указываем названия колонок.

|  |  |
| --- | --- |
| 1 | INSERT INTO months (id,name,days) VALUES (2,'February',29); |

В случае, если мы не укажем одну из колонок, на её место будет записано NULL или заданное значение по умолчанию, но это уже совсем другая история.

# Select

Данный запрос используется в случае, если нам нужно показать данные в таблице. Наверное, самым простым примером использования SELECT будет следующий запрос:

|  |  |
| --- | --- |
| 1 | SELECT \* FROM characters |

Результатом данного запроса будет таблица со всеми данными в таблице characters. Знак звёздочки (\*) означает то, что мы хотим показать все столбцы из таблицы без исключений. Так как в базе данных обычно больше одной таблицы, нам необходимо указывать название таблицы, данные из которой мы хотим посмотреть. Сделать это мы можем, используя ключевое слово FROM.

Когда вам нужны лишь некоторые столбцы из таблицы, то вы можете указать их имена через запятую вместо звёздочки.

|  |  |
| --- | --- |
| 1 | SELECT name, weapon FROM characters |

Также иногда нам нужно отсортировать выводимые данные. Для этого мы используем ORDER BY «название столбца». ORDER BY имеет два модификатора: ASC (по возрастанию) (по умолчанию) и DESC (по убыванию).

|  |  |
| --- | --- |
| 1 | SELECT name, weapon FROM "characters" ORDER BY name DESC |

# Where

Теперь мы знаем, как показать только конкретные столбцы, но что если мы хотим включить в вывод лишь некоторые конкретные строки? Для этого мы используем WHERE. Данное ключевое слово позволяет нам фильтровать данные по определённому условию.

В следующем запросе мы выведем только тех персонажей, которые в качестве оружия используют пистолет.

|  |  |
| --- | --- |
| 1  2  3 | SELECT \*  FROM characters  WHERE weapon = 'pistol'; |

# И/или

Условия в WHERE могут быть написаны с использованием логических операторов (AND/OR) и математические операторы сравнения (=, <, >, <=, >=, <>).

К примеру, у нас есть табличка, в которой записаны данные о 4 самых продаваемых музыкальных альбомах всех времён. Давайте выведем только те, жанром которых является рок, а продажи которых были меньше, чем 50 миллионов копий.

|  |  |
| --- | --- |
| 1  2  3  4 | SELECT \*  FROM albums  WHERE genre = 'rock' AND sales\_in\_millions <= 50  ORDER BY released |

# In/Between/Like

Условия в WHERE могут быть записаны с использованием ещё нескольких команд, которыми являются:

* IN — сравнивает значение в столбце с несколькими возможными значениями и возвращает true, если значение совпадает хотя бы с одним из значением
* BETWEEN — проверяет, находится ли значение в каком-то промежутке
* LIKE — ищет по шаблону

К примеру, мы можем сделать запрос для вывода данных об альбомах в жанре pop или soul:

|  |  |
| --- | --- |
| 1 | SELECT \* FROM albums WHERE genre IN ('pop','soul'); |

Если мы хотим вывести все альбомы, которые были выпущены в промежутке между 1975 и 1985 годом, мы можем использовать следующую запись:

|  |  |
| --- | --- |
| 1 | SELECT \* FROM albums WHERE released BETWEEN 1975 AND 1985; |

Также, если мы хотим вывести все альбомы, в названии которых есть буква ‘R’, мы можем использовать следующую запись:

|  |  |
| --- | --- |
| 1 | SELECT \* FROM albums WHERE album LIKE '%R%'; |

Знак % означает любую последовательность символов (0 символов тоже считается за последовательность).

Если мы хотим вывести все альбомы, первая буква в названии которых — ‘R’, то запись слегка изменится:

|  |  |
| --- | --- |
| 1 | SELECT \* FROM albums WHERE album LIKE 'R%'; |

В SQL также есть инверсия. Для примера, попробуйте самостоятельно написать NOT перед любым логическим выражением в условии (NOT BETWEEN и так далее).

# Функции

В SQL полно встроенных функций для выполнения разных операций. Мы же покажем вам только наиболее часто используемые:

* COUNT() — возвращает число строк
* SUM() — возвращает сумму всех полей с числовыми значениями в них
* AVG() — возвращает среднее значение среди строк
* MIN()/MAX() — возвращает минимальное/максимальное значение среди строк

Чтобы вывести год выпуска самого старого альбома, в таблице можно использовать следующий запрос:

|  |  |
| --- | --- |
| 1 | SELECT MAX(released) FROM albums; |

Обратите внимание, что если вы напишете запрос, в котором вам, к примеру, нужно будет вывести имя и среднее значение чего-либо, то вы получите ошибку на выводе.

Допустим, вы пишете такой запрос:

|  |  |
| --- | --- |
| 1 | SELECT name, avg(age) FROM students; |

Чтобы избежать ошибки, вам следует добавить следующую строку:

|  |  |
| --- | --- |
| 1 | GROUP BY name |

Причиной тому является, что запись avg(age) является совокупной (aggregated), и вам необходимо групировать значения по имени.

# Вложенные Select

В предыдущих шагах мы изучили, как делать простые вычисления с данными. Если мы хотим использовать результат данных вычислений, то часто нам необходимо использовать так называемые вложенные запросы. Допустим, нам необходимо вывести артиста, альбом и год выпуска самого старого альбома в таблице.

Вывести эти столбцы можно, используя следующий запрос:

|  |  |
| --- | --- |
| 1 | SELECT artist, album, released FROM albums; |

Также мы знаем, как получить самый ранний год из имеющихся:

|  |  |
| --- | --- |
| 1 | SELECT MIN(released) FROM album; |

Объеденить эти запросы можно в WHERE:

|  |  |
| --- | --- |
| 1  2  3  4  5 | SELECT artist,album,released  FROM albums  WHERE released = (   SELECT MIN(released) FROM albums  ); |

# Присоеденение таблиц

В сложных базах данных чаще всего у нас есть несколько связанных таблиц. К примеру, у нас есть две таблицы: про видеоигры и про разработчиков.

В таблице video\_games есть столбец developer\_id, в данном случае он является так называемым foreign\_key. Чтобы было проще понять, developer\_id — это связывающее звено между двумя таблицами.

Если мы хотим вывести всю информацию об игре, включая информацию о её разработчике, нам необходимо подключить вторую таблицу. Чтобы это сделать, можно использовать INNER JOIN:

|  |  |
| --- | --- |
| 1  2  3  4 | SELECT video\_games.name, video\_games.genre, game\_developers.name, game\_developers.country  FROM video\_games  INNER JOIN game\_developers  ON video\_games.developer\_id = game\_developers.id; |

Это, наверное, самый простой пример использования JOIN. Есть ещё несколько вариантов его использования. Для более подробной информации предлагаем перейти по [этой ссылке](https://www.w3schools.com/sql/sql_join.asp" \t "/home/pv/Documents\\x/_blank).

# Псевдонимы

Если вы взгляните на предыдущий пример, то вы заметите, что есть два столбца, названных одинаково: «name». Часто это может запутать. Решением данной проблемы являются псевдонимы. Они, к слову, помогают сделать название столбца красивее или понятнее в случае необходимости.

Чтобы присвоить столбцу псевдоним, можно использовать ключевое слово AS:

|  |  |
| --- | --- |
| 1  2  3  4 | SELECT games.name, games.genre, devs.name AS developer, devs.country  FROM video\_games AS games  INNER JOIN game\_developers AS devs  ON games.developer\_id = devs.id; |

# Update

Зачастую нам нужно изменить данные в таблице. В SQL это делается с помощью UPDATE.

Исполльзование UPDATE включает в себя:

* выбор таблицы, в которой находится поле, которое мы хотим изменить
* запись нового значения
* использование WHERE, чтобы обозначить конкретное место в таблице

Предположим, у нас есть таблица с самыми высокооценёнными сериалами всех времён. Однако у нас есть проблема: «Игра Престолов» обозначена как комедия и нам определённо нужно это изменить:

|  |  |
| --- | --- |
| 1  2  3 | UPDATE tv\_series  SET genre = 'drama'  WHERE name = 'Game of Thrones'; |

# Удаление записей из таблицы

Удаление записи из таблицы через SQL — очень простая операция. Всё, что нужно — это обозначить, что именно мы хотим удалить.

|  |  |
| --- | --- |
| 1  2 | DELETE FROM tv\_series  WHERE id = 4; |

Примечание: убедитесь, что используете WHERE, когда удаляете запись из таблицы. Иначе вы удалите все записи из таблицы, сами того не желая.

# Удаление таблиц

Если мы хотим удалить все данные из таблицы, но при этом оставить саму таблицу, нам следует использовать команду TRUNCATE:

|  |  |
| --- | --- |
| 1 | TRUNCATE TABLE table\_name; |

В случае, если мы хотим удалить саму таблицу, то нам следует исользовать команду DROP:

|  |  |
| --- | --- |
| 1 | DROP TABLE table\_name; |

# Заключение

На этой ноте мы завершаем данный SQL-туториал. Само собой, это не всё, и для полного освоения нужно ещё много изучить, однако данное вступление даст вам толчок для дальнейшего изучения.

Более подробные уроки по SQL вы можете найти, перейдя по следующим ссылкам:

* [Codeacademy курс SQL](https://www.codecademy.com/learn/learn-sql" \t "/home/pv/Documents\\x/_blank)
* [SQL Fiddle](http://sqlfiddle.com/" \t "/home/pv/Documents\\x/_blank): онлайн-инструмент для тестирования SQL запросов