SQL for Beginners Part 1

**CREATE DATABASE: Creating a Database**

[**https://code.tutsplus.com/articles/sql-for-beginners--net-8200**](https://code.tutsplus.com/articles/sql-for-beginners--net-8200)

VARIANTA RUSA O GASITI AICI

<http://jtest.ru/bazyi-dannyix/sql-dlya-nachinayushhix.html>

Here comes our very first query. We are going to be creating a database to work with.

First, open up your MySQL Console and login. For WAMP, the default password is blank. For MAMP, the password should be 'root' by default.

After logging in, type this query and hit enter:

|  |  |
| --- | --- |
| 1 | CREATE DATABASE my\_first\_db; |



Note that semicolon (;) is added at the end of the query, just like at the end of lines of code.

Also, the special words 'CREATE DATABASE' are case insensitive, along with all special words in SQL. But for the sake of readability, we will be writing them in uppercase.

**Optional: Character Set and Collation**

If you would like to set the default character set and collation, you can write the same query like this instead:

|  |  |
| --- | --- |
| 1 | **CREATE DATABASE my\_first\_db DEFAULT CHARACTER SET utf8 COLLATE utf8\_general\_ci;** |

**[Here](http://dev.mysql.com/doc/refman/5.1/en/charset-charsets.html)** is a list of supported character sets and collations in MySQL.

**SHOW DATABASES: List All Databases**

This query is used to get a list of all databases you have.



**DROP DATABASE: Delete a Database**

You can delete an existing database with this query.



Be careful with this query, because it gives you no warnings. If you have tables and data under the database, they will all be deleted instantly.

**USE: Selecting a Database**

This technically is not a query. It is a 'statement' and does not require a semicolon at the end.



It tells MySQL to select a default database to work with, for the current session. Now we are ready to create tables and do other things under this database.

**What is a Database Table?**

You can think of a database table like a spreadsheet or csv file that holds structured data.



Just like in this example, tables have column names, and rows of data. With SQL queries **we can create these tables**. We can also **add, read, update and delete the data**.

**CREATE TABLE: Creating a Table**

With this query we can create tables in the database. Unfortunately the [MySQL documentation](http://dev.mysql.com/doc/refman/5.1/en/create-table.html) is not very friendly for new learners. The structure of this type of query can get very complex, but we will start with an easy one.

The following query will create a table with 2 columns.

|  |  |
| --- | --- |
| 1234 | CREATE TABLE users (    username VARCHAR(20),    create\_date DATE); |

Note that we are able to write a query in multiple lines, and even use tabs for indentation.

First line is easy. We just create a table named 'users'. Following that, in parantheses, we have a list table columns separated by commas. After each column name, we have a data type, such as VARCHAR or DATE.

VARCHAR(20) means that the column is a string type, and can be a maximum of 20 characters long. DATE is also a data type that is specifically used for storing dates, in this format: 'YYYY-MM-DD'.

**PRIMARY KEY**

Before we run that query, we should also include a column for 'user\_id', which will be a PRIMARY KEY. Without getting too much into the details, you can think of a PRIMARY KEY as a way to identify each row of data in a table.

Now the query becomes:

|  |  |
| --- | --- |
| 12345 | CREATE TABLE users (    user\_id INT AUTO\_INCREMENT PRIMARY KEY,    username VARCHAR(20),    create\_date DATE); |

INT makes this a 32bit integer type (i.e. numeric). AUTO\_INCREMENT automatically generates a new id number every time we add new rows of data. It is not required, but makes it much more convenient.

This column does not have to be an integer, but it is the most commonly used type. Having a PRIMARY KEY column also is not required, but it is strongly recommended for good database design and performance.

Let's run the query:



**SHOW TABLES: List All Tables**

This query allows you to get a list of tables that are currently in the database.



**EXPLAIN: Show Table Structure**

To see the structure of an existing table, you can use this query.



Fields (aka. columns) are listed in the results, with their properties.

**DROP TABLE: Delete a Table**

Just like DROP DATABASES, this query deletes a table and its contents, without a warning.



**ALTER TABLE: Modify a Table**

This query also can have quite a [complex structure](http://dev.mysql.com/doc/refman/5.1/en/alter-table.html) because of the multitude of changes it can perform on a table. Let's look at some simple examples.

(Make sure to re-create the table we just dropped or the following queries obviously won't work.)

**Add a Column**

|  |  |
| --- | --- |
| 123 | ALTER TABLE users    ADD email VARCHAR(100)    AFTER username; |

Thanks to the readability of SQL, I don't think that query even needs an explanation.



**Remove a Column**



**That was also very simple. But use it with caution as it permanently removes data without a warning.**

Re-add the email column because we are going to be using it later:

|  |  |
| --- | --- |
| 123 | ALTER TABLE users    ADD email VARCHAR(100)    AFTER username; |

**Modify a Column**

Sometimes you may want to change the properties of a column, so you don't have to delete and recreate it.



That renamed the username column to 'user\_name' and changed the type from VARCHAR(20) to VARCHAR(30). A change like this should not disturb any of the existing data in the table.

**INSERT: Add Data to a Table**

Let's add some data into the table using this query.



As you can see, VALUES() contains the list of field values, separated by commas. The string values are enclosed in single quotes. And the values need to be in the order of the columns that were defined when we created the table.

Note that the first value is NULL for the PRIMARY KEY field we called 'user\_id'. We do this so that an id is automatically generated, because the column is set to AUTO\_INCREMENT. When entering a row of data for the first time, the id will be 1. Next inserted row will be 2 and so on...

**Alternate Syntax**

Here is another syntax for inserting rows.



This time we are using the keyword SET instead of VALUES, and it is not followed by paratheses. There are a few things to note here:

* A column can be omitted. For example we did not assign a value to user\_id, which will default to the AUTO\_INCREMENT functionality. If you omit a VARCHAR column, it would default to an empty string (unless a different default value was specified during table creation).
* Each column has to be referenced by its name. Because of this, they can be in any order, unlike the previous syntax.

**Alternate Syntax 2**

Here is yet another syntax.



Again, since each column is referenced by name, they can be in any order.

**LAST\_INSERT\_ID()**

You can use this query to get the AUTO\_INCREMENT id for the last inserted row, in the current session.



**NOW()**

I think it is a good time to demonstrate how you can use a **MySQL function** inside your queries.

The NOW() function returns the current date. So you can use it to automatically set a DATE column to the current day while inserting a new row.



Note that we received a warning from MySQL, but it is not a big deal. The reason is that NOW() actually returns time information as well.



But the create\_date column we created only contains the date, and not the time, therefor the returned data was truncated. We could use the CURDATE() function instead, which returns just the date, but the data stored at the end would be the same either way.

**SELECT: Read Data from a Table**

Obviously the data we added would be useless unless we can read it. This is where the SELECT query comes in.

Here is the simplest possible SELECT query for reading from a table:



In this case, the asterisk (\*) means that we asked to fetch all the columns from the table. If you want only specific columns, the query would look like this:



**WHERE Clause**

More often than not, we are only interested in some of the rows, and not all. For example, let's say we want the email address for the user 'nettuts'.



Think of it like an IF statement. WHERE allows you to put conditions in the query for the results you are looking for.

Note that for the equality condition, only a single equal sign is used (=), instead of double (==) which you might be used to from programming.

You can use other comparison conditions too:



AND and OR can be used to combine conditions:



Note that numeric values do not have to be inside quotes.

**IN()**

This is useful for matching multiple values.



**LIKE**

This allows you to do wildcard searches.



Percentage sign (%) is used as the wildcard.

**ORDER BY Clause**

If you want the results to be returned in a specific order, use this clause:



The default order is ASC (i.e. ascending). You can add DESC to reverse order it.

**LIMIT ... OFFSET ...**

You can limit the number of returned results.



LIMIT 2 just gets the first 2 results. LIMIT 1 OFFSET 2 gets 1 result, after the first 2 results. LIMIT 2, 1 means the same thing, but note that the first number is the offset and the second number is the limit.

**UPDATE: Update Data in a Table**

This query is used for updating the data in a table.



Most of the time, it is used with a WHERE clause, because you would want only specific rows to be updated. If a WHERE clause is not provided, all rows would be updated with the same changes.

You can also use a LIMIT clause to limit the number of rows to be updated.



**DELETE: Delete Data from a Table**

Just like UPDATE, this query is also usually used with a WHERE clause.



**TRUNCATE TABLE**

To delete the entire contents of a table, you can just do this:

|  |  |
| --- | --- |
| 1 | DELETE FROM users; |

But it is usually more performance efficient to use TRUNCATE instead.



TRUNCATE also resets the AUTO\_INCREMENT numbers so a new row will again have the id 1. But this does not happen with a DELETE query, and the counter keeps going up.

Advertisement

**Escaping String Values and Special Words**

**String Values**

Certain characters need to be escaped, otherwise you can have problems.



Backslash (\) is used for escaping.

This is also very important for security reasons. Any user input going into the database needs to be properly escaped. In PHP, you use the [mysql\_real\_escape\_string()](http://php.net/manual/en/function.mysql-real-escape-string.php) function or use prepared statements since they do escaping automatically.

**Special Words**

Since MySQL has many special words like SELECT or UPDATE, you can prevent collision by putting quotes around your table and column names. But these are not the regular quotes; you need to use the backtick (`) character.

Let's say you want to add a column named 'delete' for some reason:



**Conclusion**

Thank you for reading the article. Hopefully I was able to show you that SQL has an easy learning curve yet it is very powerful.