Tutorial 1 - MySQL Tutorial for Beginners

MySQL is a popular relational database management system that is used for storing and managing data. This tutorial will guide you through the basics of MySQL, covering installation, basic commands, and fundamental concepts.

Table of Contents – Tutorial 1

Table of Contents – Tutorial 2

Table of Contents – Tutorial 3

Installation

To install MySQL, follow these steps:

1. **Download MySQL Community Server** from the official MySQL website.
2. **Run the installer** and follow the installation wizard instructions.
3. **Configure MySQL** (set root password, select server type, etc.).
4. **Start the MySQL server**.

Connecting to MySQL

You can connect to MySQL using the command line or a GUI client like MySQL Workbench.

Using Command Line

```
mysql -u root -p
```

- -u root: specifies the username (root by default).
- -p: prompts for your password.

Using MySQL Workbench

1. Open MySQL Workbench.
2. Click on the **+** icon to create a new connection.
3. Enter your connection details and click **Test Connection**.

Basic Commands

Creating a Database

To create a new database, use the following command:

Syntax:

```
CREATE DATABASE database_name;
```

Example:

```
CREATE DATABASE college_db;
```

Creating a Table

To create a table, use the following syntax:

```
CREATE TABLE table_name (
  column1_name column1_datatype,
  column2_name column2_datatype,
  ...
);
```

Example:

```
CREATE TABLE users (
  id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(100),
  email VARCHAR(100)
);
```

Inserting Data

To insert data into a table, use:

```
INSERT INTO table_name (column1, column2, ...) VALUES (value1, value2, ...);
```

Example:

```
INSERT INTO users (name, email) VALUES ('John Doe', 'john@example.com');
```

Querying Data

To retrieve data from a table, use:

```
SELECT * FROM table_name;
```

Example:

```
SELECT * FROM users;
```

Updating Data

To update existing data in a table, use:

Syntax:

UPDATE table_name SET column1 = value1, column2 = value2 WHERE condition;

Example:

UPDATE users SET email = 'john.doe@example.com' WHERE name = 'John Doe';

Deleting Data

To delete data from a table, use:

DELETE FROM table_name WHERE condition;

Example:

DELETE FROM users WHERE name = 'John Doe';

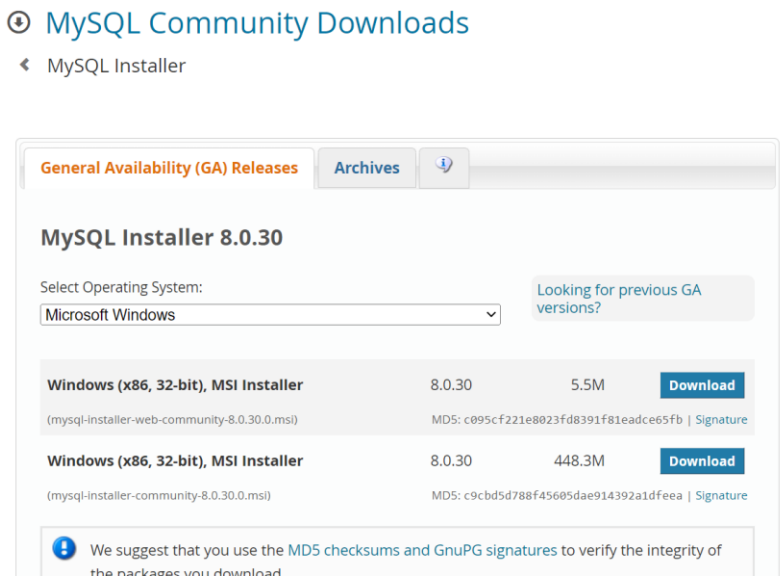
Conclusion

This tutorial covered the fundamentals of MySQL, including installation, connecting to the database, and basic commands for managing data. As you become more familiar with MySQL, you can explore advanced topics such as indexing, joins, and stored procedures

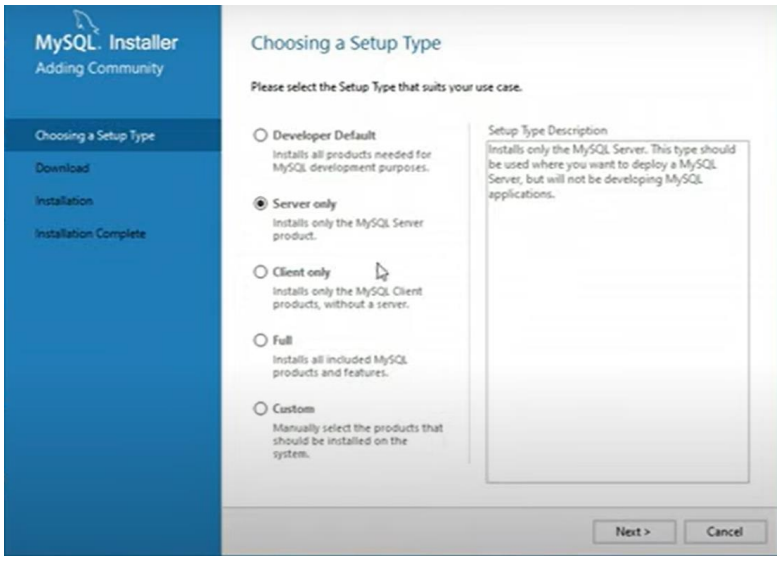
Learning for Life...

Tutorial 2 - Installing MySQL

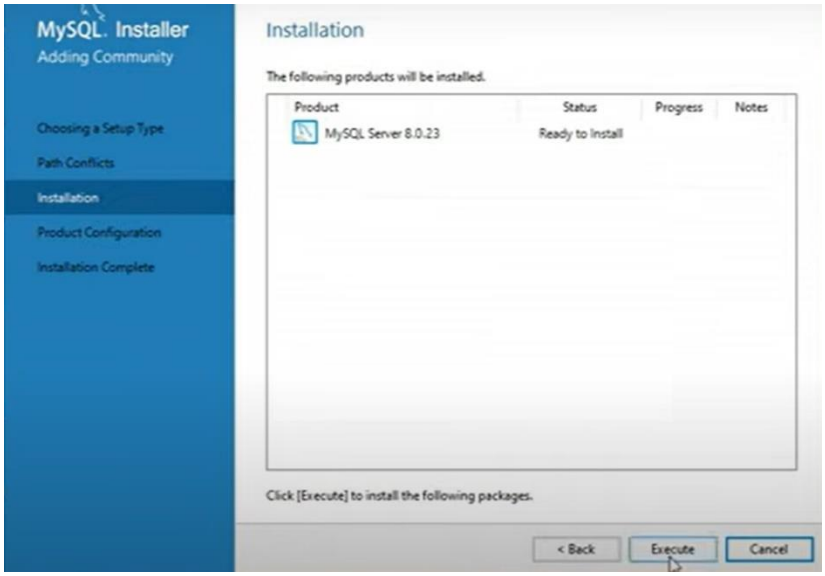
1/ Download the MySQL Community Edition Server installer from [MySQL :: Download MySQL Installer]



2/ Accept the License Terms and select “Server only”

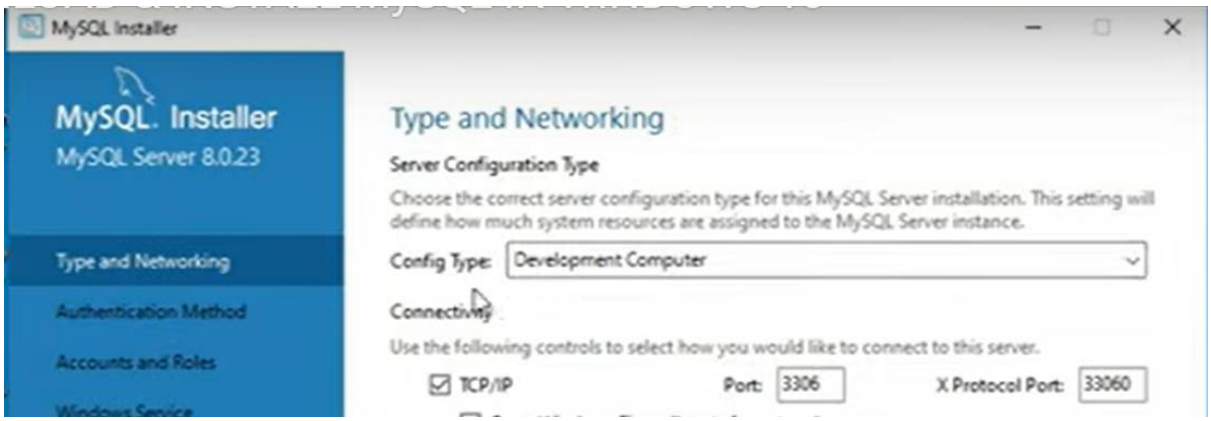


3/ Click the Execute button to install



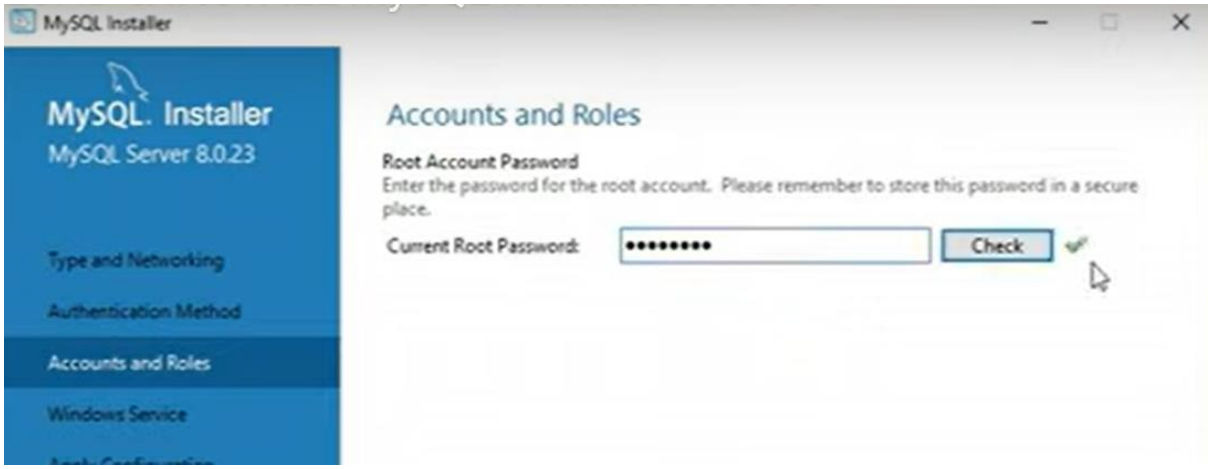
4/ If asked, choose the “Standalone MySQL Server” option and Next to continue
(I didn’t get this option)

5/ Select “Development Computer” as the server configuration



6/ Select “Use Strong Password Encryption for Authentication” then click Next

7/ Enter (Twice if asked to repeat) a Root User password
– KEEP A SEPARATE NOTE OF THIS PASSWORD



8/ Have you written your password down? – IF NOT DO IT NOW!

9/ Click Next to run as a Windows Service, Next to ignore Plugins and Execute to install

Exercise

Launch MySQL Command Line Client from the group in the Start menu

Use your root password to login

Enter password:**********

This should give you a prompt

```
mysql>
```

BEFORE WE PROCEED – Note that MySQL is case sensitive and if you are not confident It might be better to copy and paste your commands from this document. Your Word Doc may hide underscore characters under a red squiggly line as it thinks they are a spelling error.

Type the following SQL Statement carefully – make sure to include the semi-colon

```
CREATE DATABASE IF NOT EXISTS web_site_db;
```

Try the following – remember the semi-colon at the end of each statement

```
SHOW DATABASES;
```

Now we will create a database user with your own first name and a password of "pa55word" – Nb: Be careful to only add the semi-colon at the end of the whole statement ie: not each line

```
CREATE USER IF NOT EXISTS 'graeme'@'localhost'
IDENTIFIED WITH mysql_native_password BY 'pa55word';
```

Now we will grant our user some privileges

```
GRANT SELECT, INSERT, UPDATE ON web_site_db.*
TO 'graeme'@'localhost';
```

Now try

```
SHOW GRANTS FOR 'graeme'@'localhost';
```

Okay – Let's create a simple table in our database
Run the following statement

```
CREATE TABLE IF NOT EXISTS users (
user_id INT UNSIGNED NOT NULL AUTO INCREMENT,
fname VARCHAR(20) NOT NULL,
lname VARCHAR(20) NOT NULL,
```

```
eml VARCHAR(50) NOT NULL,
PRIMARY KEY(user_id),
UNIQUE(eml));
```

Now let's check that has worked
Try the following

```
EXPLAIN users;
```

You should now see the details of your table

user_id INT UNSIGNED NOT NULL AUTO INCREMENT, - means that the user_id field holds non negative whole numbers, the field must have a value and the numbers will increment or count up by 1 each time.

VARCHAR(20) is a field that takes a 20 character string

NOT NULL means the field must have a value

PRIMARY KEY(user_id) means that user_id field has been configured as the unique identifier field for the table

UNIQUE(eml) means that this field must also hold unique values

Conclusion
Congratulations if you have reached this far, we now have a basic relational database with one table waiting for us to populate it with some data.

Tutorial 3 - MySQL **Join Feature Tutorial**

In this tutorial, we'll explore the **JOIN** feature in MySQL using two sample tables: **students** and **courses**. We'll cover the different types of joins and provide example queries for each type.

Sample Tables

Students Table

student_id,	name, age,	age	course_id
1	Alice	20	101
2	Bob	21	102
3	Charlie	22	101
4	David	23	NULL

Courses Table

course_id	course_name
101	Mathematics
102	Science
103	Literature

Types of Joins

An **INNER JOIN** returns records that have matching values in both tables.

Example:

```
SELECT s.name, c.course_name
FROM students s
INNER JOIN courses c ON s.course_id = c.course_id;
```

Result:

Name	Course_name
Alice	Mathematics
Bob	Science
Charlie	Mathematics

A **LEFT JOIN** returns all records from the left table (students), and the matched records from the right table (courses). If there is no match, NULL values are returned for columns from the right table.

Example:

```
SELECT s.name, c.course_name
FROM students s
LEFT JOIN courses c ON s.course_id = c.course_id;
```

Result:

name	course_name
Alice	**Mathematics**
Bob	**Science**
Charlie	**Mathematics**
David	**Null**

A **RIGHT JOIN** returns all records from the right table (courses), and the matched records from the left table (students). If there is no match, NULL values are returned for columns from the left table.

Example:

```
SELECT s.name, c.course_name
FROM students s
RIGHT JOIN courses c ON s.course_id = c.course_id;
```

Result:

name	course_name
Alice	**Mathematics**
Bob	**Science**
Charlie	**Mathematics**
NULL	**Literature**

MySQL does not directly support **FULL OUTER JOIN**, but you can simulate it using a combination of **LEFT JOIN** and **RIGHT JOIN** with a **UNION**.

Example:

```
SELECT s.name, c.course_name
FROM students s
LEFT JOIN courses c ON s.course_id = c.course_id

UNION

SELECT s.name, c.course_name
FROM students s
RIGHT JOIN courses c ON s.course_id = c.course_id;
```

Result:

name	Course_name
Alice	Mathematics
Bob	Science
Charlie	Mathematics
David	NULL
NULL	Literature

Conclusion

In this tutorial, we've covered the basic types of **JOIN** in MySQL using **students** and **courses** tables. Understanding these joins is crucial for effectively querying relational databases.