

# Java SE & Spring

## Module 1: Java SE

## 16. Java Database Connectivity (JDBC)



# JDBC

The Java Database Connectivity (JDBC) API provides universal data access from the Java. Using the JDBC API, you can access virtually any data source, from relational databases to spreadsheets and flat files. The JDBC API is comprised of two packages:

- `java.sql`
- `javax.sql`

Both packages are downloaded with Java Platform Standard Edition (Java SE).

---

Further reading: <https://howtodoinjava.com/jdbc-tutorials/>

# JDBC Drivers

Java database connectivity (JDBC) is JavaSoft specification of a standard application programming interface (API) that allows Java programs to access database management systems. The JDBC API consists of a set of interfaces and classes written in the Java programming language. Using these standard interfaces and classes, programmers can write applications that connect to databases, send queries written in structured query language (SQL), and process the results. JDBC is oriented towards relational databases.

Because JDBC is a standard specification, a Java program that uses the JDBC API can connect to any database management system (DBMS) for which there is a JDBC driver.

---

Further reading: <https://howtodoinjava.com/jdbc-tutorials/>

# JDBC Drivers

The JDBC API defines the Java interfaces and classes that programmers use to connect to databases and send queries.

A Java program (that uses the JDBC API) loads the specified driver for a particular DBMS before it actually connects to a database. The JDBC's DriverManager class then sends all JDBC API calls to the loaded driver.

There are 4 different types of JDBC drivers:

- Type 1 : JDBC-ODBC bridge driver
- Type 2 : Native-API Driver
- Type 3 : All Java + Middleware translation driver
- Type 4 : Pure Java driver

---

Further reading: <https://howtodoinjava.com/jdbc-tutorials/>

# Database Types

Generally, databases is comprised of two types. First, Relational Databases (SQL), and Not Relational (NoSQL) databases. The most common types of NoSQL databases are key-value, document, column and graph databases.

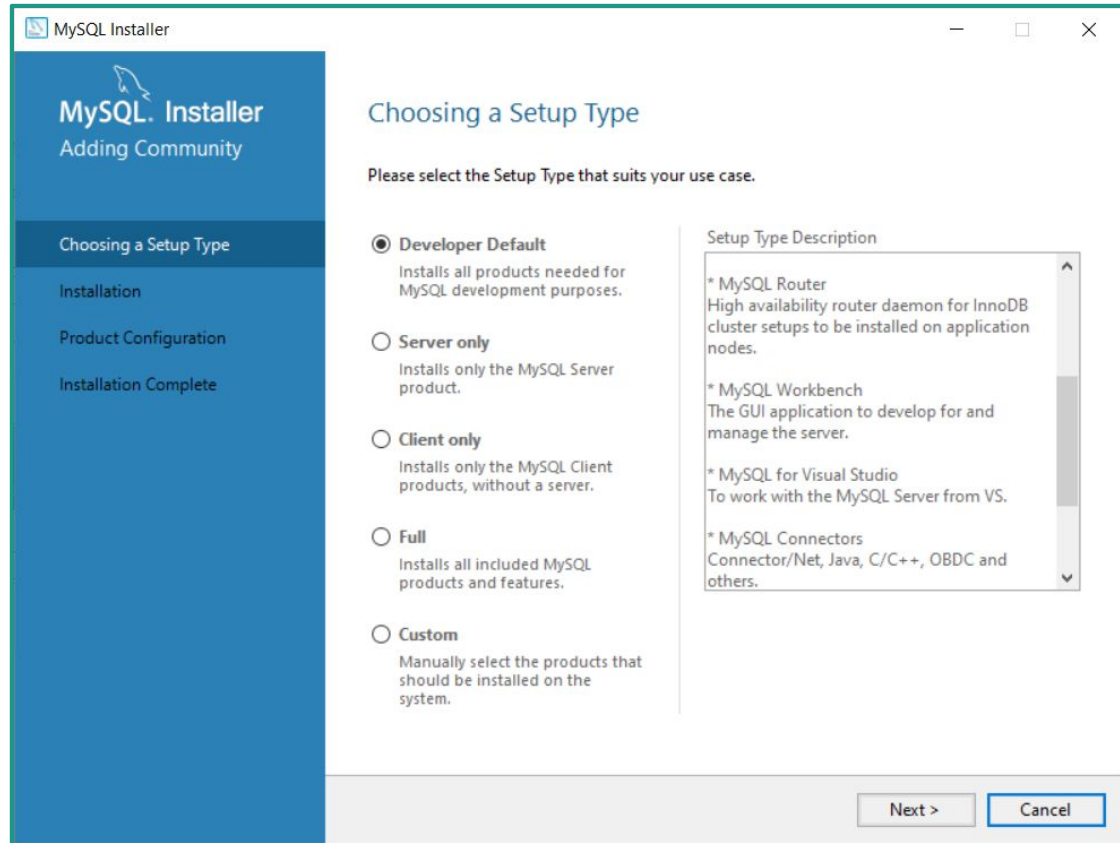
In this course we will talk about MySQL Relational Database. It is open source and has a free community version. You can download it from [here](#).

# Installing MySQL

After you download the MySQL installer from the link at the previous slide.

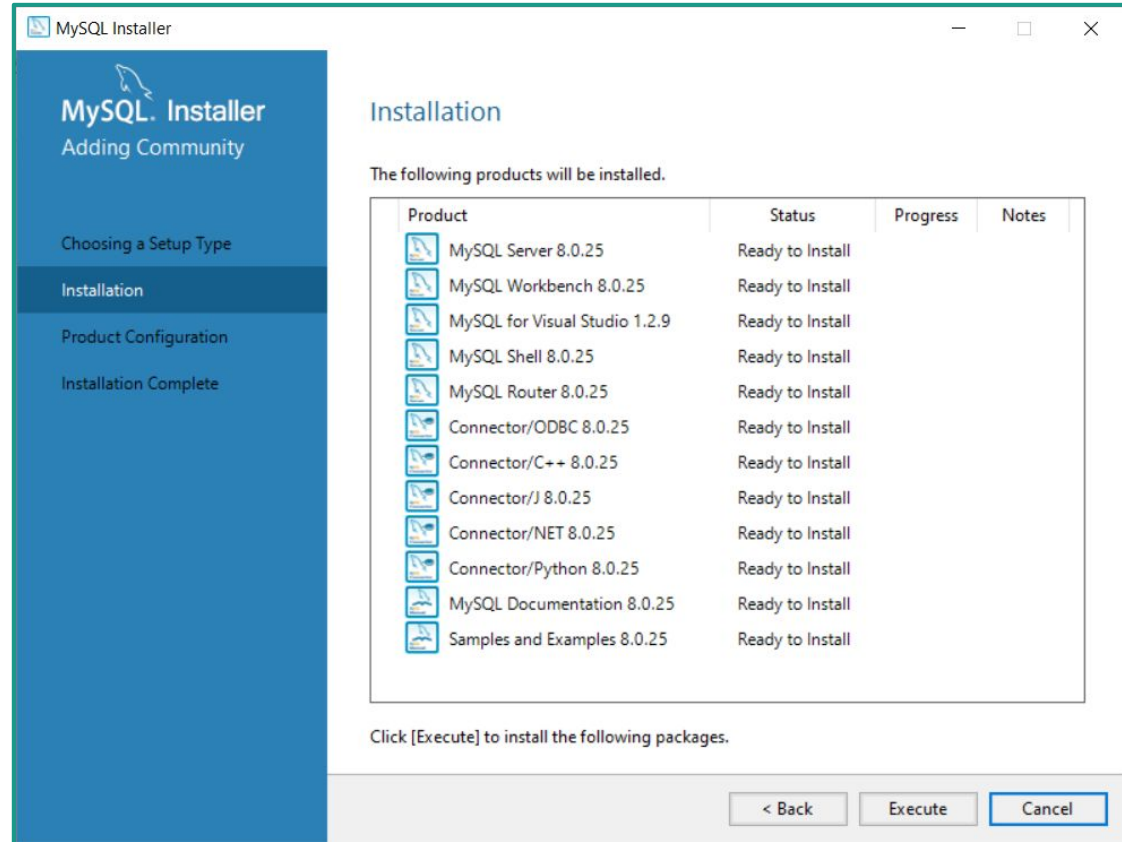
Open the installer by double-clicking ➡

Choose “Developer Default”, and click **Next**



# Installing MySQL

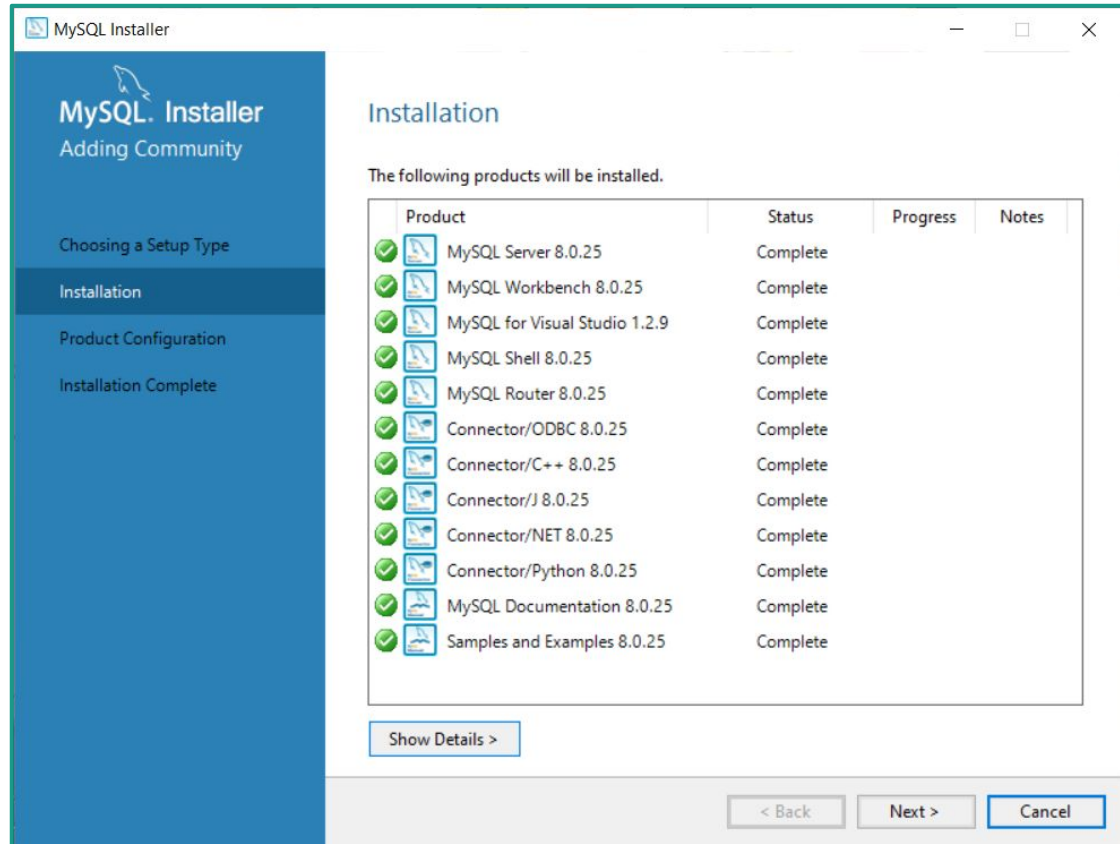
Click the **Execute** button.





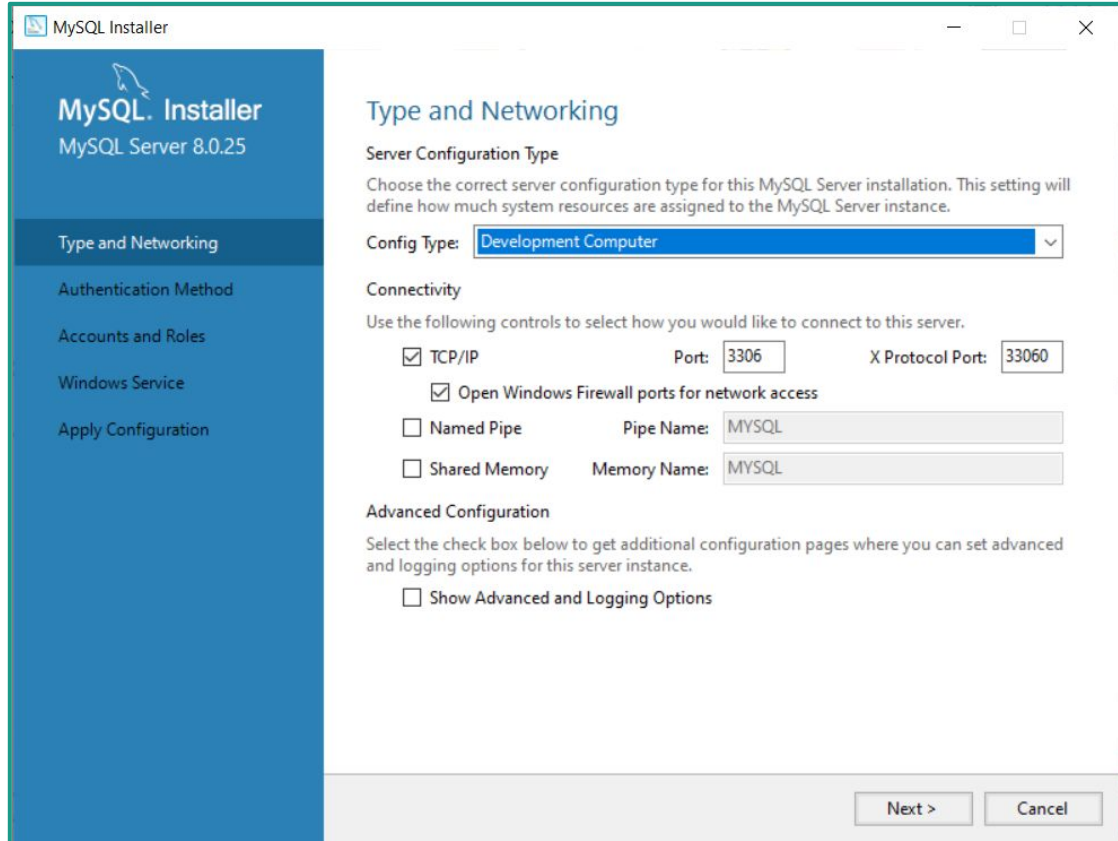
# Installing MySQL

When installation is completed ,  
Click **Next** button .



# Installing MySQL

In the configuration page,  
choose “Development  
Computer” as config type and  
click **Next**.



The screenshot shows the MySQL Installer window for MySQL Server 8.0.25. The left sidebar contains the following options: Type and Networking (selected), Authentication Method, Accounts and Roles, Windows Service, and Apply Configuration. The main area is titled 'Type and Networking' and includes the following sections:

- Server Configuration Type**: A description states, 'Choose the correct server configuration type for this MySQL Server installation. This setting will define how much system resources are assigned to the MySQL Server instance.' The 'Config Type' dropdown menu is set to 'Development Computer'.
- Connectivity**: A description states, 'Use the following controls to select how you would like to connect to this server.' The options are:
  - ☒ TCP/IP: Port is 3306, X Protocol Port is 33060.
  - ☒ Open Windows Firewall ports for network access.
  - ☐ Named Pipe: Pipe Name is MYSQL.
  - ☐ Shared Memory: Memory Name is MYSQL.
- Advanced Configuration**: A description states, 'Select the check box below to get additional configuration pages where you can set advanced and logging options for this server instance.' There is an unchecked checkbox for 'Show Advanced and Logging Options'.

At the bottom right, there are 'Next >' and 'Cancel' buttons.

# Installing MySQL

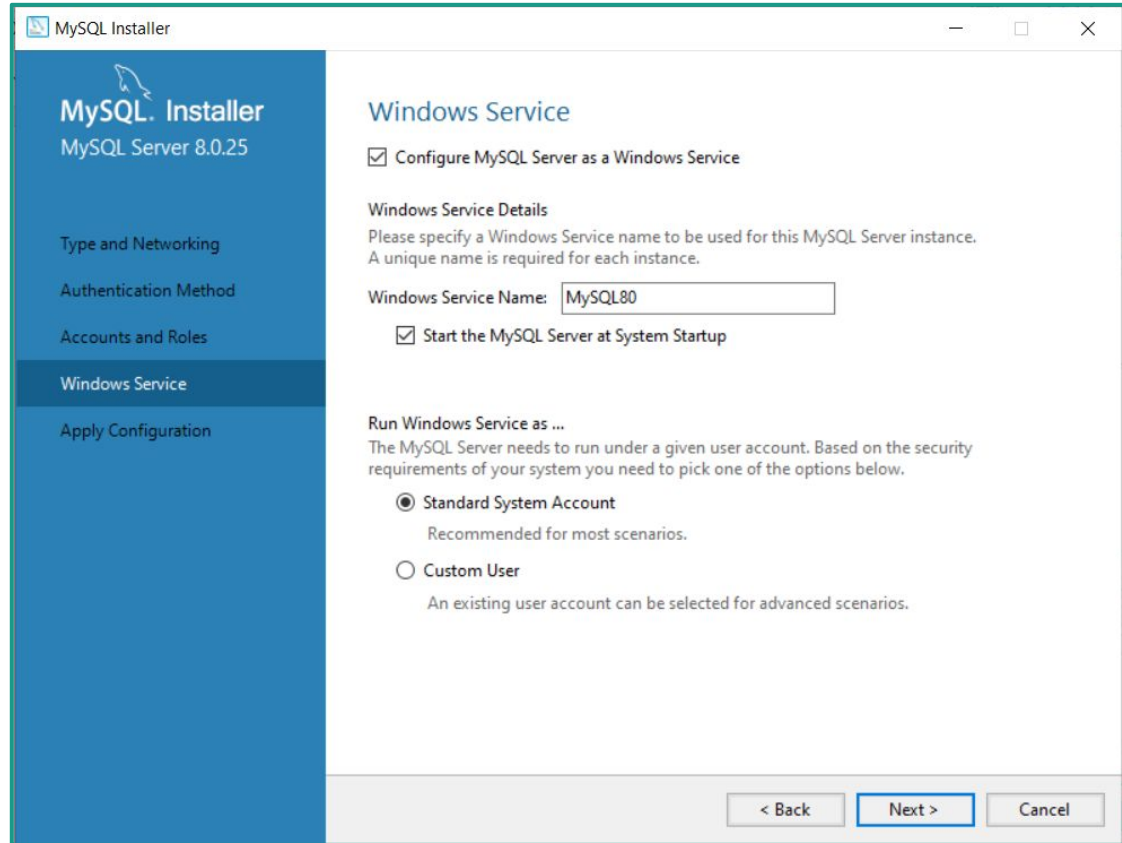
Define root user password and  
create a user with DBA role.  
(Strong, and hard to guess by  
others passwords are  
recommended. 😊)

The screenshot shows the 'MySQL Installer' window for 'MySQL Server 8.0.25'. The left sidebar has five options: 'Type and Networking', 'Authentication Method', 'Accounts and Roles' (which is selected and highlighted in dark blue), 'Windows Service', and 'Apply Configuration'. The main area is titled 'Accounts and Roles' and contains two sections. The first section, 'Root Account Password', asks for the root password and includes two input fields: 'MySQL Root Password:' and 'Repeat Password:'. Both fields are filled with dots. Below them, it says 'Password strength: Strong' in green. The second section, 'MySQL User Accounts', has a description: 'Create MySQL user accounts for your users and applications. Assign a role to the user that consists of a set of privileges.' Below this is a table with three columns: 'MySQL User Name', 'Host', and 'User Role'. There is one row with a user icon, the name 'JAVACOURSE', the host 'localhost', and the role 'DB Admin'. To the right of the table are three buttons: 'Add User', 'Edit User', and 'Delete'. At the bottom of the window are three buttons: '< Back', 'Next >', and 'Cancel'.

MySQL User Name	Host	User Role
JAVACOURSE	localhost	DB Admin

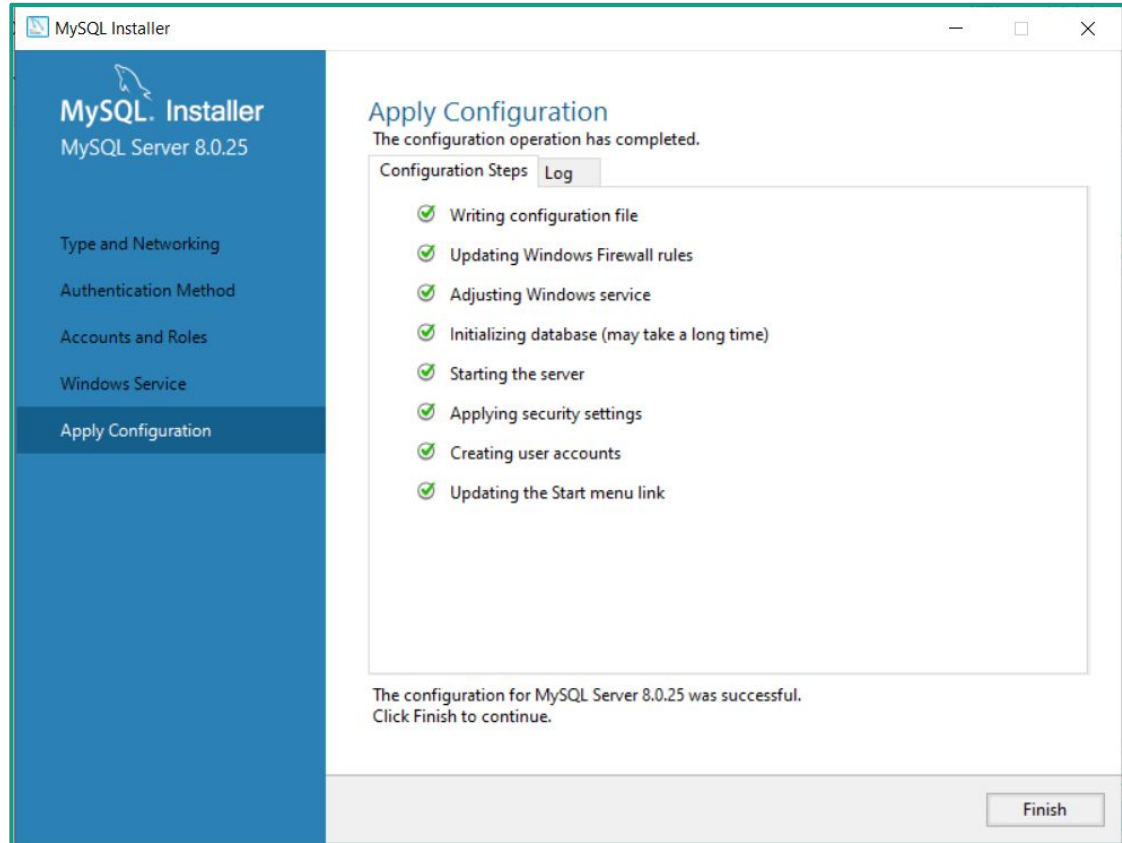
# Installing MySQL

Leave the options as it is in this page and click **Next**.



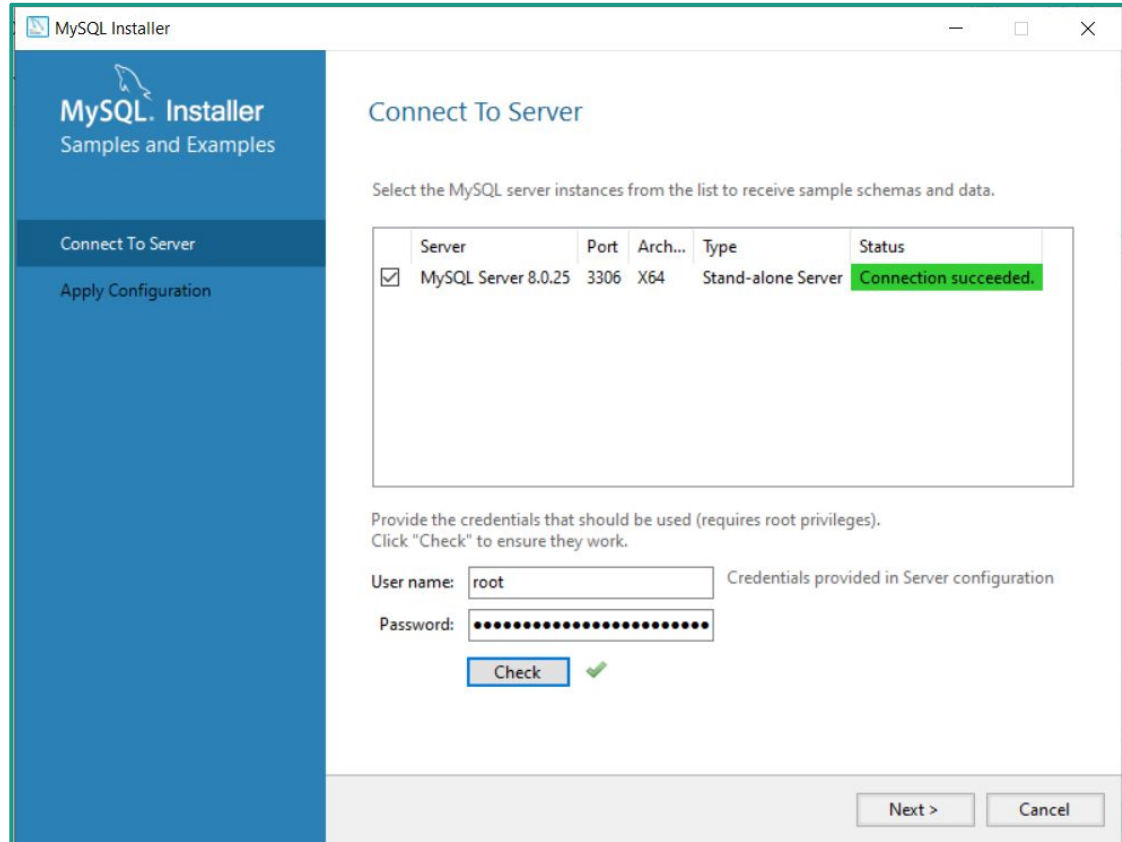
# Installing MySQL

Then, click the execute button, after the configuration is finished, you should see this page. Click the **Finish** button to continue remaining steps.



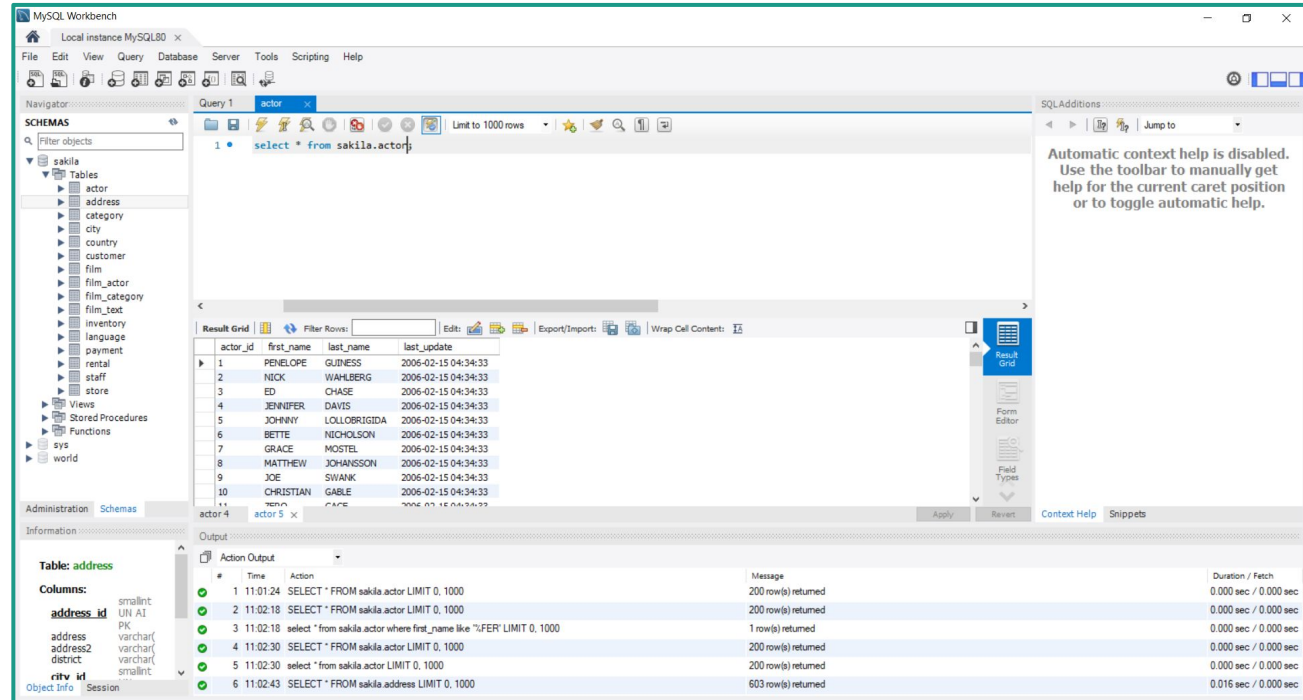
# Installing MySQL

After clicking **Next** button a several time, you will receive to connection test page. Provide the root password you defined in previous steps and click the **Check** button to test you are able to connect to the DB.



# Installing MySQL

When installation is finished,  
MySQL Workbench  
automatically will be opened.  
You should see a window like  
this. ➡  
This means we successfully  
finished the MySQL installation.



# Java – JDBC Connection

Handling a connection requires following steps:

- 1) Load JDBC driver
- 2) Open DB connection
- 3) Close DB connection



# Load JDBC Driver

First, we need to download MySQL JDBC driver. You can download it [here](#). The easiest way to do this is to use **Class.forName()** on the class that implements the `java.sql.Driver` interface. With MySQL Connector/J, the name of this class is **com.mysql.jdbc.Driver**. With this method, you could use an external configuration file to supply the driver class name and driver parameters to use when connecting to a database.

```
Class.forName("com.mysql.cj.jdbc.Driver");
```

# Load JDBC Driver

The easiest way to do this is to use **Class.forName()** on the class that implements the `java.sql.Driver` interface. With MySQL Connector/J, the name of this class is **com.mysql.jdbc.Driver**. With this method, you could use an external configuration file to supply the driver class name and driver parameters to use when connecting to a database.

```
Class.forName("com.mysql.cj.jdbc.Driver");
```

## Example:

```
1 try {
2     Class.forName("com.mysql.cj.jdbc.Driver");
3     System.out.println("Driver is found successfully");
4 }
5 catch (ClassNotFoundException e){
6     System.out.println("The JDBC driver for MySQL not found!");
7     return;
8 }
```

# Open DB Connection

After the driver has been registered with the DriverManager, you can obtain a Connection instance that is connected to a particular database by calling **DriverManager.getConnection()**. This method automatically loads JDBC drivers in the classpath.

## Syntax:

```
Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/", "root", "password");
```

---

Further reading: <https://docs.oracle.com/javase/tutorial/jdbc/basics/connecting.html>

# Open DB Connection

## Example:

```
1 Connection mySqlConnection = null;
2 try {
3     mySqlConnection = DriverManager.getConnection("jdbc:mysql://localhost:3306/"
4 + schema, "root", "password");
5     System.out.printf("Successfully connected to the database and schema: %s",
6 schema);
7 } catch (SQLException e) {
8     System.out.println("Unable to connect to the DB at the specified
9 address!");
10    return;
}
```

---

Further reading: <https://docs.oracle.com/javase/tutorial/jdbc/basics/connecting.html>

# JDBC Query Examples

SQL SELECT, INSERT, UPDATE or DELETE queries are executed to fetch, add, change or remove data stored in relational databases. It requires following steps:

- 1) Make a database connection
- 2) Execute the SQL Query
- 3) Fetch the data from result set

# JDBC Select Query Examples

As a main step, it requires creating a Statement object and then using it's **executeQuery()** method.

```
1 Statement statement = mySqlConnection.createStatement();
2 ResultSet selectResultSet = statement.executeQuery("SELECT * FROM CITY WHERE
3 CITY_ID <= 10");
4
5 while (selectResultSet.next()){
6     System.out.println("-----");
7     System.out.println("City Id: " + selectResultSet.getString(1));
8     System.out.println("City: " + selectResultSet.getString(2));
9     System.out.println("Country Id: " + selectResultSet.getString(3));
10    System.out.println("Last Update Date: " + selectResultSet.getString(4));
11 }
```

You can use various **getXXX()** (such as **getInt()**, **getDate()**, **getBoolean()** etc.) methods available in **ResultSet**. But if you want to make it generic, then use **getString()** method and parse the data as and when needed.

# JDBC Insert Example

## Example:

```
1 Connection connection = null;
2 Statement stmt = null;
3 try{
4     Class.forName("com.mysql.jdbc.Driver");
5     connection = DriverManager
6         .getConnection("jdbc:mysql://localhost:3306/TestSchema", "root",
7     "password");
8
9     stmt = connection.createStatement();
10    stmt.execute("INSERT INTO EMPLOYEE (ID,FIRST_NAME, LAST_NAME, AGE) "
11                + "VALUES (1, 'Ahmet', 'Yilmaz', 5)");
12 }
13 catch(SQLException e){
14     e.printStackTrace();
15 }
```

# JDBC Update Example

## Example:

```
1 String sql = "UPDATE CITY SET CITY = 'Metro City', LAST_UPDATE = '2021-07-16' +  
2 \"14:40:58' WHERE CITY_ID = 999\";  
3     ResultSet rs = stmt.executeQuery(QUERY);  
4     while(rs.next()){  
5         //Display values  
6         System.out.print("ID: " + rs.getInt("id"));  
7         System.out.print(", Age: " + rs.getInt("age"));  
8         System.out.print(", First: " + rs.getString("first"));  
9         System.out.println(", Last: " + rs.getString("last"));  
10    }  
11
```



# JDBC Delete Example

## Example:

```
1 connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/JDBCDemo",  
2 "root", "password");  
3  
4 stmt = connection.createStatement();  
5 stmt.execute("DELETE FROM EMPLOYEE WHERE ID >= 1");
```

# JDBC Prepared Statements

In database management systems, a prepared statement or parameterized statement is a feature used to execute the same or similar database statements repeatedly with high efficiency. Typically used with SQL statements such as queries or updates, the prepared statement takes the form of a template into which certain constant values are substituted during each execution.

# Advantages of Prepared Statements

- Pre-compilation and DB-side caching of the SQL statement leads to overall faster execution and the ability to reuse the same SQL statement in batches.
- Automatic prevention of SQL injection attacks by built-in escaping of quotes and other special characters. Note that this requires that you use any of the PreparedStatement setXxx() methods to set the values and not use inline the values in the SQL string by string concatenation.
- Apart from above two main usage, prepared statements makes it easy to work with complex objects like BLOBs and CLOBs.

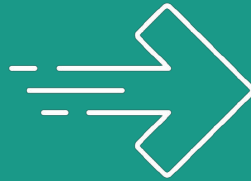
# Advantages of Prepared Statements

## Example:

```
1 PreparedStatement pstmt = connection.prepareStatement(sql);  
2 pstmt.setInt(1, 87);  
3 pstmt.setString(2, "Ahmet");  
4 pstmt.setString(3, "Yılmaz");  
5 pstmt.setInt(4, 5);  
6 pstmt= preparedStatement.executeQuery();
```



# Questions ?



## Next:Maven Build Tool