Java Servlet and JDBC Example | Insert data in MySQL

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 10 Dec, 2019

**Prerequisites:** [Servlet](https://www.geeksforgeeks.org/introduction-java-servlets/), [JDBC Connectivity](https://www.geeksforgeeks.org/establishing-jdbc-connection-in-java/)

**To start with interfacing Java Servlet Program with JDBC Connection:**

1. Proper JDBC Environment should set-up along with database creation.
2. To do so, download the mysql-connector.jar file from the internet,
3. As it is downloaded, move the jar file to the apache-tomcat server folder,
4. Place the file in **lib** folder present in the apache-tomcat directory.
5. **To start with the basic concept of interfacing:**
   * **Step 1: Creation of Database and Table in MySQL**

As soon as jar file is placed in the folder, create a database and table in MySQL,

mysql> create database demoprj;

Query OK, 1 row affected (4.10 sec)

mysql> use demoprj

Database changed

mysql> create table demo(id int(10), string varchar(20));

Query OK, 0 rows affected (1.93 sec)

mysql> desc demo;

+--------+-------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+--------+-------------+------+-----+---------+-------+

| id | int(10) | YES | | NULL | |

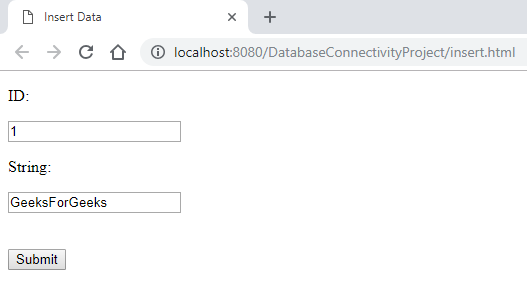
| string | varchar(20) | YES | | NULL | |

+--------+-------------+------+-----+---------+-------+

2 rows in set (0.40 sec)

* + **Step 2: Implementation of required Web-pages**  
    Create a form in HTML file, where take all the inputs required to insert data into the database. Specify the servlet name in it, with the POST method as security is important aspects in database connectivity.

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title>Insert Data</title>  </head>  <body>      <!-- Give Servlet reference to the form as an instances      GET and POST services can be according to the problem statement-->      <form action="./InsertData" method="post">          <p>ID:</p>          <!-- Create an element with mandatory name attribute,          so that data can be transfer to the servlet using getParameter() -->          <input type="text" name="id"/>          <br/>          <p>String:</p>          <input type="text" name="string"/>          <br/><br/><br/>          <input type="submit"/>      </form>  </body>  </html> |

* + **Output:**  
    
  + Submit the data (with validation) as all the required data are inserted.
  + **Step 3: Creation of Java Servlet program with JDBC Connection**

To create a JDBC Connection steps are

* + 1. Import all the packages
    2. Register the JDBC Driver
    3. Open a connection
    4. Execute the query, and retrieve the result
    5. Clean up the JDBC Environment

Create a separate class to create a connection of database, as it is a lame process to writing the same code snippet in all the program. Create a .java file which returns a Connection object.

|  |
| --- |
| import java.sql.Connection;  import java.sql.DriverManager;  import java.sql.SQLException;    // This class can be used to initialize the database connection  public class DatabaseConnection {      protected static Connection initializeDatabase()          throws SQLException, ClassNotFoundException      {          // Initialize all the information regarding          // Database Connection          String dbDriver = "com.mysql.jdbc.Driver";          String dbURL = "jdbc:[mysql://](NULL) localhost:3306/";          // Database name to access          String dbName = "demoprj";          String dbUsername = "root";          String dbPassword = "root";            Class.forName(dbDriver);          Connection con = DriverManager.getConnection(dbURL + dbName,                                                       dbUsername,                                                       dbPassword);          return con;      }  } |

* + **Step 4: To use this class method, create an object in Java Servlet program**

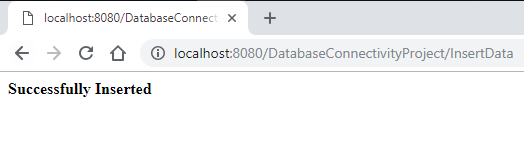
Below program shows Servlet Class which create a connection and insert the data in the **demo** table,

|  |
| --- |
| import java.io.IOException;  import java.io.PrintWriter;  import java.sql.Connection;  import java.sql.PreparedStatement;    import javax.servlet.ServletException;  import javax.servlet.annotation.WebServlet;  import javax.servlet.http.HttpServlet;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;    // Import Database Connection Class file  import code.DatabaseConnection;    // Servlet Name  @WebServlet("/InsertData")  public class InsertData extends HttpServlet {      private static final long serialVersionUID = 1L;        protected void doPost(HttpServletRequest request,  HttpServletResponse response)          throws ServletException, IOException      {          try {                // Initialize the database              Connection con = DatabaseConnection.initializeDatabase();                // Create a SQL query to insert data into demo table              // demo table consists of two columns, so two '?' is used              PreparedStatement st = con                     .prepareStatement("insert into demo values(?, ?)");                // For the first parameter,              // get the data using request object              // sets the data to st pointer              st.setInt(1, Integer.valueOf(request.getParameter("id")));                // Same for second parameter              st.setString(2, request.getParameter("string"));                // Execute the insert command using executeUpdate()              // to make changes in database              st.executeUpdate();                // Close all the connections              st.close();              con.close();                // Get a writer pointer              // to display the successful result              PrintWriter out = response.getWriter();              out.println("<html><body><b>Successfully Inserted"                          + "</b></body></html>");          }          catch (Exception e) {              e.printStackTrace();          }      }  } |

* + **Step 5: Get the data from the HTML file**

To get the data from the HTML file, the request object is used which calls [getParameter()](https://www.geeksforgeeks.org/getparameter-passing-data-from-client-to-jsp/) Method to fetch the data from the channel. After successful insertion, the writer object is created to display a success message.

After insertion operation from Servlet, data will be reflected in MySQL Database

**Output:**  


**Result in MySQL Interface**

mysql> select \* from demo;

+------+---------------+

| id | string |

+------+---------------+

| 1 | GeeksForGeeks |

+------+---------------+

1 row in set (0.06 sec)

This article shows the basic connection of JDBC with Java Servlet Program, to insert data in large volume then proper validation should be done as if any data which is not in proper format will encounter an error. All the data inserting in Database should be encrypted.

**JDBC in MYSQL in Servlet**

[<< Servlet Annotation](https://codedec.com/tutorials/annotations-in-servlet/)

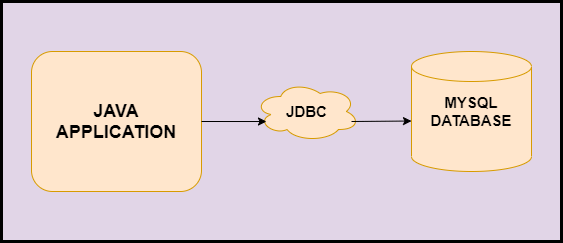
[Oracle Database Connectivity >>](https://codedec.com/tutorials/how-to-connect-oracle-database-with-servlet/)

In this tutorial, I am going to cover Database connectivity between **Java and MYSQL with the servlet.** Whatever the data we enter in input boxes it needs to be store at some location(Server or local computer) that is called a database.

There are many database tools available in the market but the most demanding tools are  ***MYSQL, Oracle, MongoDB, SQLite***ETC. In this **Servlet tutorial,** I Will cover the All the Java database connectivity examples with **MYSQL.**

**What is JDBC(Java Database Connectivity)?**

**JDBC** is a **Java API** that is used to create connectivity to the Java code and Database. ***java.sql***package contains all the classes, interfaces, and methods that provide support to manage a database with a Java application.



**DriverManager Class**

DriverManager class acts as an interface between users and drivers. It established a connection between the database and the appropriate driver. It calls the following methods:

* **registerDriver(Driver driver):** It is used to register the given driver with DriverManager.
* **deregisterDriver(Driver driver):** It is used to deregister the driver.
* **getConnection(String url):** It is used to established connection with the URL.
* **getConnection(String url, String name, String pass):** It is used to established connection with the URL, specified database name, database password. It returns the object of Connection.

**Connection Interface**

A connection is an interface to keep the session between java application & database. It provides some methods:

* **createSatement():** This is used to create a statement object that is used to execute SQL queries.
* **createStatement(int resultSetType,int resultSetConcurrency):**This create the statement object and generate Resultset type with concurrency.
* **setAutoCommit():** This method sets the commit status.
* **commit():** This method saves the changes made.
* **rollback():** This method drops all the changes made since the previous commit.
* **close():** This method closes the connection.

**Statement Interface**

It is an interface that provides methods to execute queries with the database. It is a factory of ResultSet.

* **executeQuery(String sql):** This method is used to execute a SELECT query that returns the Resultset object.
* **executeUpdate(String sql):** This method is used to execute a DML query.
* **execute(String sql):** This method returns multiple results.
* **executeBatch():** This method is used to execute the Batch of commands.

1. Statement st=con.**createStatement**();
2. int query=st.**executeQuery**("INSERT INTO TABLE\_DEMO VALUES("ABC",21)");

**PreparedStatement Interface**

This interface is used to execute parameterized queries. Let’s look at the methods of it.

* **setInt(int index,int values):** This method sets the Integer values to given index.
* **setString(int index, string value):** This method sets the string value to the given index.
* **setFloat(int index, float value):** This method sets the float value to the given index.
* **setDouble(int index, double value):** This method sets the double value to the given index.
* **executeUpdate():** This method is used for executing queries.
* **executeQuery():** This method executes the SELECT query.

1. PreparedStatement pstmt=con.**prepareStatement**("insert into TABLE\_DEMO values(?,?)");
2. pstmt.**setString**(1,"Nicolas");
3. pstmt.**setInt**(2,21);

**Difference between Statement and PreparedStatement**

|  |  |
| --- | --- |
| **Statement** | **PreparedStatement** |
| It is used for executing a static SQL statement in JDBC. | It is used for executing pre-compiled SQL statement in JDBC. |
| It cannot accept parameters at runtime. | It can accept different parameters. |
| It is slower as compared to PreparedStatement. | It is faster as compared to the statement because it executes precompiled SQL statements. |
| It is suitable for executing DDL commands. | It is suitable for executing DML commands |
| It makes code less readable and understandable. | It makes code more readable and understandable. |
| It is present in java.sql.Statement | It is present in java.sql.PreparedStatement |

**ResultSet Interface**

This interface object is pointed to the row of a table. The ResultSet interface provides methods for retrieving column values from the current row. Let’s see the methods in ResultSet Interface.

* **next():** This method is used to move the cursor one location at a time.
* **previous():** This method is used to move to the previous location from the current position.
* **first():** This is used to move the cursor to the first row.
* **last():** This method is used to move to the last row.
* **absolute(int row):** This method is used to move the cursor to a specified row number.
* **relative(int row):** This method is used to move the cursor to a relative row number.
* **getInt(int column\_index):** This method is used to return integer data from the specified column index number.
* **getInt(String column\_name):** This method is used to return integer data from the specified column name.
* **getString(int column\_index):** This method is used to return string data from the specified column index.
* **getString(String column\_name):** This method is used to return string data from the specified column name.

1. ResultSet rs=st.**executeQuery**("SELECT \* FROM TABLE\_DEMO");
2. **while**(rs.**next**())
3. {
4. rs.**getString**("name"); //get the column values.
5. rs.**getInt**("RollNo");
6. }

**ResultSetMetaData Interface**

ResultSetMetadata is used to get the metadata of the data(extra information about data) like the total number of rows, rows name rows data type, etc. Let’s see the methods:

* **int getColumnCount():**This method returns total number of columns.
* **String getColumnName():** This method returns Column Name.
* **String getTableName():** This method returns the table name.

1. ResultSetMetaData rs=rs.**getMetaData**();
2. System.out.**pritnln**("Column Name of index 2 :"+rs.**getColumnName**(2));

**DatabaseMetaData Interface**

DatabaseMetaData interface is used to get the metadata of the database like database name, driver name, etc. Let’s see some of the methods in this interface.

* **String getDriverName():** This method returns the database driver’s name.
* **String getDriverVersion():** This method returns the version of Driver.
* **String getUserName():** This method returns the username of the database.
* **String getDatabaseProductName():** This method returns the product name of the database.

1. DatabaseMetadata st=con.**getMetaData**();
2. System.out.**pritnln**("Driver Name :"+st.**getDriverName**());

**JDBC Connectivity with MYSQL**

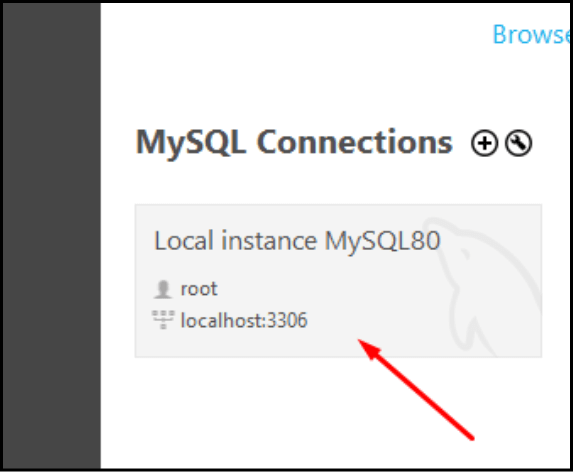
Mysql is a free, open-source RDBMS(Relational Database Management System). MYSQL is one of the best RDBMS being used for developing various web-based software applications. We will look at the steps to install MYSQL in Windows OS.

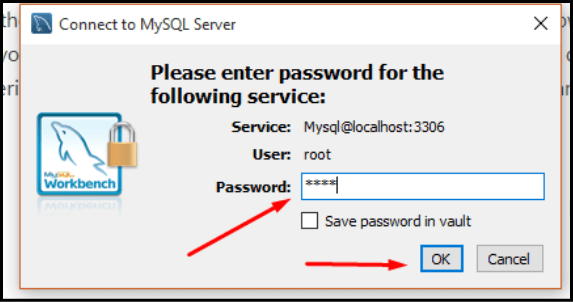
Download the MySQL database from <https://dev.mysql.com/downloads/mysql/> After Downloading the MYSQL community version, double click the installer file. Now the Installer window will be launched. Now, Follow the below steps.

1. Select the Full radio button to install MYSQL. Click on Next.
2. Click on Execute. It will take time to install the package (have patience here)
3. Click on Next. again Click on Next.
4. Now, you will see Type and Networking Page. Click Next.
5. Here it will ask you to write username and password which is important so make sure you keep it simple as it will be good to remember (I will suggest you keep both username and password as **root**). Click on Next.
6. Now, on the Windows Service page check the above two options of Configure MySQL server and start the MYSQL and then select the Standart system Account radio button and click Next.
7. Now, When the configuration will be finished Click Finish to continue.

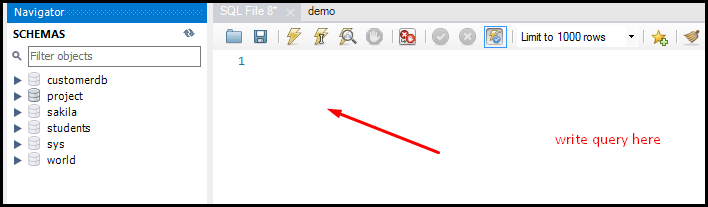
**Step 1: Create a Database and Table in MYSQL.**

We can create a database using MySQL Workbench. Go to the Windows start menu and search MySQL workbench and click to open it. Now Select the First MySQL Instance.

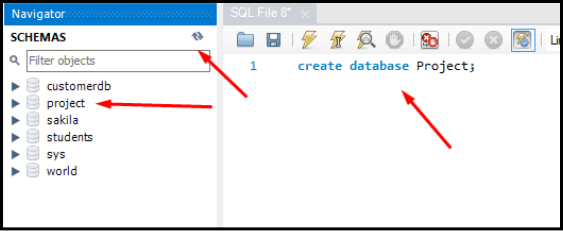




**After that, we can access the MYSQL query browser run it perform an operation, etc.**



**Write the query in the MYSQL Query tab then refresh the schemas you will get the database that you have created.**



Now,  create a table, write  the following query in the Query tab, and execute it.

1. use Project
2. create table project.**demo**(id **int**(20), string **varchar**(20));

Now Table will be created in the database having two column **Id** and **String**.

**Step 2: Create a project in eclipse (as we normally create it) add the jar file for mysql-connector.jar(open directory of MySql where you have installed >open Connector >copy mysql-connector-java8.0.21 into lib folder)**

**Step 3: Create an HTML file to take the required input & insert it into the database.**

**index.html**

1. <form action="JDBCExample" method="post">
2. Enter Id<input type="text" name="id"><br>
3. Enter Name<input type="text" name="name">
4. <br>
5. <input type="submit" value="InsertIntoDB" id="button-1"/>
6. </form>

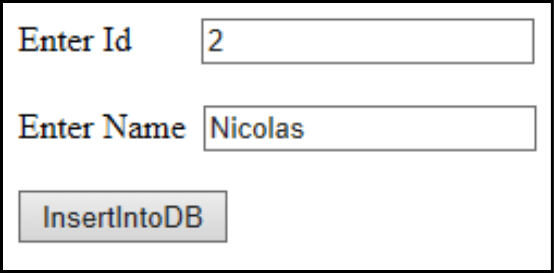
**Step 4: Create a class for initializing the database**

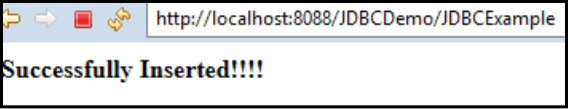
1. public class DatabaseCon {
2. protected static Connection **getConn**() throws SQLException, ClassNotFoundException {
3. String dbDriver = "com.mysql.jdbc.Driver";
4. String dbURL = "jdbc:mysql://localhost:3306/Project";
5. String dbUsername = "root";
6. String dbPassword = "root";
7. Class.**forName**(dbDriver);
8. Connection con = DriverManager.**getConnection**(dbURL, dbUsername, dbPassword);
9. return con;
10. }
11. }

**Step 5: Create a Servlet and insert the data into the database**

1. protected void **doPost**(HttpServletRequest request, HttpServletResponse response)
2. throws ServletException, IOException {
3. // TODO Auto-generated method stub
4. int id = Integer.**parseInt**(request.**getParameter**("id"));
5. String name = request.**getParameter**("name");
6. try {
7. Connection connection = DatabaseCon.**getConn**();
8. PreparedStatement ps = connection.**prepareStatement**("insert into demo values(?,?)");
9. ps.**setInt**(1, id);
10. ps.**setString**(2, name);
11. ps.**executeUpdate**();
12. ps.**close**();
13. PrintWriter out = response.**getWriter**();
14. out.**println**("<html><body><b>Successfully Inserted!!!!" + "</b></body></html>");
15. } **catch** (ClassNotFoundException e) {
16. // TODO Auto-generated catch block
17. e.**printStackTrace**();
18. } **catch** (SQLException e) {
19. // TODO Auto-generated catch block
20. e.**printStackTrace**();
21. }
22. }
23. }

**Output**





Now view the database by executing the query **select \* from demo** to check whether the data is inserted or not.

|  |  |
| --- | --- |
| Id | String |
| 2 | Nicolas |

Now data will be stored in the database. Thus this is how we connect ***MYSQL Database*** in Servlet.