Java Database Connectivity with MySQL

To connect Java application with the MySQL database, we need to follow 5 following steps.

In this example we are using MySql as the database. So we need to know following informations for the mysql database:

1. **Driver class:**The driver class for the mysql database is **com.mysql.jdbc.Driver**.
2. **Connection URL:**The connection URL for the mysql database is **jdbc:mysql://localhost:3306/Test** where jdbc is the API, mysql is the database, localhost is the server name on which mysql is running, we may also use IP address, 3306 is the **port number** and Test is the database name. We may use any database, in such case, we need to replace the Test with our database name.
3. **Username:**The default username for the mysql database is **root**.
4. **Password:**It is the password given by the user at the time of installing the mysql database. In this example, we are going to use root as the password also we may leave it as blank.

Let's first create a table in the mysql database, but before creating table, we need to create database first.

1. create database Test;
2. create table emp(id **int**(10),name varchar(40),age **int**(3));

# Java Database Connectivity with 5 Steps

|  |
| --- |
| There are 5 steps to connect any java application with the database using JDBC. These steps are as follows:   * Register the Driver class * Create connection * Create statement * Execute queries * Close connection |

1) Register the driver class

|  |
| --- |
| The **forName()** method of Class class is used to register the driver class. This method is used to dynamically load the driver class. |

Syntax of forName() method

1. **public** **static** **void** forName(String className)**throws** ClassNotFoundException

### Example to register the OracleDriver driver class

Here, Java program is loading oracle driver to esteblish database connection.

Class.forName("oracle.jdbc.driver.OracleDriver");

### Example to register the mysql driver class

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

2) Create the connection object

|  |
| --- |
| The **getConnection()** method of DriverManager class is used to establish connection with the database. |

Syntax of getConnection() method

1) **public** **static** Connection getConnection(String url)**throws** SQLException

2) **public** **static** Connection getConnection(String url,String name,String password)   **throws** SQLException

Example to establish connection with the Oracle database

Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","password");

Example to establish connection with the JDBC database

Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/Test","root","root");

//here Test is database name, root is username and password

3) Create the Statement object

|  |
| --- |
| The createStatement() method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database. |

Syntax of createStatement() method

1. **public** Statement createStatement()**throws** SQLException

Example to create the statement object

1. Statement stmt=con.createStatement();

4) Execute the query

|  |
| --- |
| The executeQuery() method of Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the records of a table. |

Syntax of executeQuery() method

1. **public** ResultSet executeQuery(String sql)**throws** SQLException

Example to execute query

ResultSet rs=stmt.executeQuery("select \* from emp");

**while**(rs.next()){

System.out.println(rs.getInt(1)+" "+rs.getString(2) )+" "+rs.getString(3));

}

5) Close the connection object

|  |
| --- |
| By closing connection object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection. |

Syntax of close() method

1. **public** **void** close()**throws** SQLException

Example to close connection

1. con.close();

Example to Connect Java Application with mysql database

In this example, Test is the database name, root is the username and password both.

**import** java.sql.\*;

**class** MysqlCon{

**public** **static** **void** main(String args[]){

**try**{

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

Connection con=DriverManager.getConnection(

"jdbc:mysql://localhost:3306/Test","root","root");

//here Test is database name, root is username and password

Statement stmt=con.createStatement();

ResultSet rs=stmt.executeQuery("select \* from emp");

**while**(rs.next())  {

System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));  }

con.close();

}**catch**(Exception e){ System.out.println(e);}

}

}

The above example will fetch all the records of emp table.

Output:

id: 101 name: pooja age: 21

id: 102 name: chandani age: 21

id: 103 name: purushottam age: 21

id: 104 name: jitendra age: 22

To connect java application with the mysql database, **mysqlconnector.jar** file is required to be loaded.

[download the jar file mysql-connector.jar](https://static.javatpoint.com/src/jdbc/mysql-connector.jar)

Two ways to load the jar file:

1. Paste the mysqlconnector.jar file in jre/lib/ext folder
2. Set classpath

1) Paste the mysqlconnector.jar file in JRE/lib/ext folder:

|  |
| --- |
| Download the mysqlconnector.jar file. Go to jre/lib/ext folder and paste the jar file here. |

2) Set classpath:

|  |
| --- |
| There are two ways to set the classpath:   * temporary * permanent |

How to set the temporary classpath

|  |
| --- |
| open command prompt and write: |

1. C:>set classpath=c:\folder\mysql-connector-java-5.0.8-bin.jar;.;

How to set the permanent classpath

Go to environment variable then click on new tab. In variable name write **classpath** and in variable value paste the path to the mysqlconnector.jar file by appending mysqlconnector.jar;.; as C:\folder\mysql-connector-java-5.0.8-bin.jar;.;

# Simplifying CRUD Operation with JDBC

**Creating, reading, updating, and deleting** data in a database is a common task in many applications, and [JDBC](https://www.geeksforgeeks.org/introduction-to-jdbc/) (Java Database Connectivity) is a Java API that allows you to connect to a database and perform these operations

**1. Connect to the database**

The first step is to establish a connection to the database. You can do this by loading the JDBC driver and creating a connection object.

* Java

|  |
| --- |
| try {      Class.forName("com.mysql.jdbc.Driver");      Connection con = DriverManager.getConnection(          "jdbc:<mysql://localhost:3306/mydb>", "username",          "password");  /\*Class.*forName*("com.mysql.cj.jdbc.Driver");  Connection con=DriverManager.*getConnection*(  "jdbc:mysql://localhost:3306/test","root","");  \*/      System.out.println("Connection established.");  }  catch (Exception e) {      e.printStackTrace();  } |

**2. Create a new record**

Once you have a connection to the database, you can use the connection object to create a new record in the database. To do this, you will need to use an SQL INSERT statement and execute it using the connection object.

* Java

|  |
| --- |
| try {     String sql = "INSERT INTO table\_name (column1, column2, column3) VALUES (?, ?, ?)";     PreparedStatement statement = con.prepareStatement(sql);     statement.setString(1, "value1");     statement.setString(2, "value2");     statement.setInt(3, 123);     statement.executeUpdate();     System.out.println("Record created.");  } catch (SQLException e) {     e.printStackTrace();  } |

**3. Read a record**

To read a record from the database, you will need to use an SQL SELECT statement and execute it using the connection object. The result of the query will be a ResultSet object that you can use to access the data in the record.

* Java

|  |
| --- |
| try {     String sql = "SELECT column1, column2, column3 FROM table\_name WHERE id = ?";     PreparedStatement statement = con.prepareStatement(sql);     statement.setInt(1, 1);     ResultSet result = statement.executeQuery();     if (result.next()) {         String column1 = result.getString("column1");         String column2 = result.getString("column2");         int column3 = result.getInt("column3");         System.out.println("Column 1: " + column1);         System.out.println("Column 2: " + column2);         System.out.println("Column 3: " + column3);     }  } catch (SQLException e) {     e.printStackTrace();  } |

**4. Update a record**

To update a record in the database, you will need to use an SQL UPDATE statement and execute it using the connection object.

* Java

|  |
| --- |
| try {     String sql = "UPDATE table\_name SET column1 = ?, column2 = ?, column3 = ? WHERE id = ?";     PreparedStatement statement = con.prepareStatement(sql);     statement.setString(1, "new\_value1");     statement.setString(2, "new\_value2");     statement.setInt(3, 456);     statement.setInt(4, 1);     statement.executeUpdate();     System.out.println("Record updated.");  } catch (SQLException e) {     e.printStackTrace();  } |

**5. Delete a record**

To delete a record from the database, you will need to use an SQL DELETE statement and execute it using the connection object.

* Java

|  |
| --- |
| try {     String sql = "DELETE FROM table\_name WHERE id = ?";     PreparedStatement statement = con.prepareStatement(sql);     statement.setInt(1, 1);     statement.executeUpdate();     System.out.println("Record deleted.");  } catch (SQLException e) {     e.printStackTrace();  } |