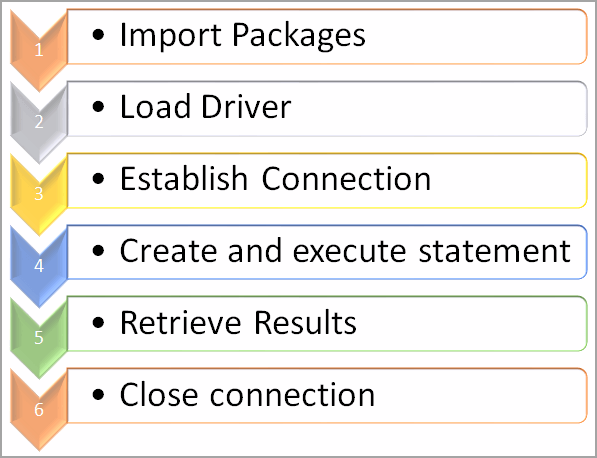
**JDBC Connection Steps**

**There are 6 basic steps to connect with JDBC. They are enlisted in the below image:**

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2020/06/2020-06-08_10-36-13.png)

**#1) Import Packages**

First, we need to import the existing packages to use it in our Java program. Import will make sure that JDBC API classes are available for the program. We can then use the classes and subclasses of the packages.

**Irrespective of the JDBC Driver, add the following import statement in the Java program.**

import java.sql.\*;

Import the other classes based on the functionality which you will use in the program. Download the appropriate Jar files for the database which you will use in the program.

*Refer to the*[***previous tutorial***](https://www.softwaretestinghelp.com/jdbc-tutorial/)*for the links to download the Jar files for your database.*

**JDBC API 4.0 mainly provides 2 important packages:**

* java.sql
* javax.sql

**(i) java.sql package**

This package provides classes and interfaces to perform most of the JDBC functions like creating and executing SQL queries.

| **Classes/ Interfaces** | **Description** |
| --- | --- |
| **BLOB** | It represents SQL Blob value in Java program |
| **CallableStatement** | It is used to execute SQL stored procedures |
| **CLOB** | It represents SQL Clob value in Java program |
| **Connection** | It creates a connection (session) with a specific Database |
| **Date** | It provides support for Date SQL type |
| **Driver** | It creates an instance of a Driver with Driver Manager |
| **DriverManager** | It provides basic service to manage a set of JDBC Drivers |
| **ParameterMetaData** | It is an object which can be used to get the information about the types and properties of each parameter in a PreparedStatement Object |
| **PreparedStatement** | It is used to create and execute a parameterized query in the Java program |
| **ResultSet** | It is used to access the result row-by-row |
| **ResultSetMetaData** | It is used to get the information about the types and properties of the columns in a ResultSet object |
| **RowId** | It represents the SQL ROWID value |
| **Savepoint** | It represents savepoint in transaction |
| **SQLData** | It is used to map the SQL User Defined Type (UDT) to a class in Java program |
| **SQLXML** | It represents SQL XML type |
| **Statement** | It is used to execute a static SQL statement |
| **DriverPropertyInfo** | It provides Driver properties to make a connection |
| **SQLException** | It provides information on database errors |
| **SQLTimeoutException** | It is a subclass of SQLException thrown when the timeout specified by the statement has expired |
| **SQLWarning** | It is an exception that provides information on database access warnings |
| **Struct** | It is a standard mapping in Java program for SQL structured type |

**(ii) javax.sql package**

It is a JDBC extension API and provides server-side data access and processing in Java Program.

| **Classes/ Interfaces** | **Description** |
| --- | --- |
| **CommonDataSource** | It is an interface that defines the methods which are common between DataSource, XADataSource and ConnectionPoolDataSource |
| **ConnectionPoolDataSource** | It is a factory for PooledConnection objects |
| **DataSource** | It is a factory for connections to the physical DataSource that the object represents |
| **PooledConnection** | It is used to manage Connection Pool |
| **RowSet** | It provides support to the JDBC API for Java beans Component Model |
| **RowSetMetadata** | It has the information about the columns in a RowSet object |
| **ConnectionEvent** | It provides information about the occurrence of connection-related events |
| **ConnectionEventListener** | It is used to register PooledConnection object events |
| **RowSetEvent** | It generates when an event occurs to a Rowset object |
| **StatementEvent** | It is sent to all StatementEventListeners which were registered with a PooledConnection generated |

**#2) Load Driver**

First, we should load/register the driver in the program before connecting to the Database. You need to register it only once per database in the program.

We can load the driver in the following 2 ways:

1. **Class.forName()**
2. **DriverManager.registerDriver()**

(i) Class.forName()

In this way, the driver’s class file loads into the memory at runtime. It implicitly loads the driver. While loading, the driver will register with JDBC automatically.

| **DB Name** | **JDBC Driver Name** |
| --- | --- |
| **MySQL** | com.mysql.jdbc.Driver |
| **Oracle** | oracle.jdbc.driver.OracleDriver |
| **Microsoft SQL Server** | com.microsoft.sqlserver.jdbc.SQLServerDriver |
| **MS Access** | net.ucanaccess.jdbc.UcanaccessDriver |
| **PostgreSQL** | org.postgresql.Driver |
| **IBM DB2** | com.ibm.db2.jdbc.net.DB2Driver |
| **Sybase** | com.sybase.jdbcSybDriver |
| **TeraData** | com.teradata.jdbc.TeraDriver |

**Note:** forName() method is valid only for JDK Compliant Virtual Machines.

(ii) DriverManager.registerDriver()

DriverManager is an inbuilt class that is available in the java.sql package. It acts as a mediator between Java application and database which you want to connect. Before you connect with the database, you need to register the driver with DriverManager. The main function of DriverManager is to load the driver class of the Database and create a connection with DB.

**Public static void registerDriver(driver)** – This method will register the driver with the Driver Manager. If the driver is already registered, then it won’t take any action.

* It will throw **SQLException** if the database error occurs.
* It will throw **NullPointerException** if the driver is null.

DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver())

DriverManager.registerDriver(new com.microsoft.sqlserver.jdbc.SQLServerDriver())

Like this, you can register the driver for your Database by passing it as a parameter.

**#3) Establish Connection**

After loading the driver, the next step is to create and establish the connection. Once required, packages are imported and drivers are loaded and registered, then we can go for establishing a Database connection.

DriverManager class has the getConnection method, we will use this method to get the connection with Database. To call getConnection() method, we need to pass 3 parameters. The 3 parameters are string data type URL, a username, and a password to access the database.

**The getConnection() method is an overloaded method. The 2 methods are:**

* **getConnection(URL,username,password);** – It has 3 parameters URL, username, password.
* **getConnection(URL);** – It has only one parameter. URL has a username and password also.

**The following table lists the JDBC connection strings for the different databases:**

| **Database** | **Connection String/DB URL** |
| --- | --- |
| **MySQL** | jdbc:mysql://HOST\_NAME:PORT/DATABASE\_NAME |
| **Oracle** | jdbc:oracle:thin:@HOST\_NAME:PORT:SERVICE\_NAME |
| **Microsoft SQL Server** | jdbc:sqlserver://HOST\_NAME:PORT;DatabaseName=< DATABASE\_NAME> |
| **MS Access** | jdbc:ucanaccess://DATABASE\_PATH |
| **PostgreSQL** | jdbc:postgresql://HOST\_NAME:PORT/DATABASE\_NAME |
| **IBM DB2** | jdbc:db2://HOSTNAME:PORT/DATABASE\_NAME |
| **Sybase** | jdbc:Sybase:Tds:HOSTNAME:PORT/DATABASE\_NAME |
| **TeraData** | jdbc:teradata://HOSTNAME/database=< DATABASE\_NAME>,tmode=ANSI,charset=UTF8 |

**Example:**

Connection con = DriverManager.getConnection(jdbc:oracle:thin:@localhost:1521:xe,System,Pass123@)

**Here in this example,**

* **thin** refers to the Driver type.
* **localhost** is where the Oracle database is running.
* **1521** is the port number to connect to DB.
* **xe** – SID
* **System** – User name to connect to the Oracle Database.
* **Pass123@** – Password

**#4) Create And Execute Statement**

Once the connection has established, we can interact with the connected Database. First, we need to create the statement to perform the SQL query and then execute the statement.

(i) Create Statement

Now we will create the statement object that runs the query with the connected database. We use the createStatement method of the ***Connection*** class to create the query.

**There are 3 statement interfaces are available in the java.sql package. These are explained below:**

**a) Statement**

This interface is used to implement simple SQL statements with no parameter. It returns the ResultSet object.

Statement statemnt1 = conn.createStatement();

**b) PreparedStatement**

This PreparedStatement interface extends the Statement interface. So, it has more features than the Statement interface. It is used to implement parameterized and precompiled SQL statements. The performance of the application increases because it compiles the query only once.

It is easy to reuse this interface with a new parameter. It supports the IN parameter. Even we can use this statement without any parameter.

String select\_query = “Select \* from states where state\_id = 1”;

PreparedStatement prpstmt = conn.prepareStatement(select\_query);

**c) CallableStatement**

CallableStatement interface extends the PreparedStatement interface. So, it has more features than the PreparedStatement interface. It is used to implement a parameterized SQL statement that invokes procedure or function in the database. A stored procedure works like a method or function in a class. It supports the IN and OUT parameters.

The CallableStatement instance is created by calling the prepareCall method of the Connection object.

CallableStatementcallStmt = con.prepareCall("{call procedures(?,?)}");

(ii) Execute The Query

**There are 4 important methods to execute the query in Statement interface. These are explained below:**

* ResultSet executeQuery(String sql)
* int executeUpdate(String sql)
* boolean execute(String sql)
* int []executeBatch()

**a) ResultSet executeQuery(String sql)**

The executeQuery() method in Statement interface is used to execute the SQL query and retrieve the values from DB. It returns the ResultSet object. Normally, we will use this method for the SELECT query.

**b) executeUpdate(String sql)**

The executeUpdate() method is used to execute value specified queries like INSERT, UPDATE, DELETE (DML statements), or DDL statements that return nothing. Mostly, we will use this method for inserting and updating.

**c) execute(String sql)**

The execute() method is used to execute the SQL query. It returns **true** if it executes the SELECT query. And, it returns **false** if it executes INSERT or UPDATE query.

**d) executeBatch()**

This method is used to execute a batch of SQL queries to the Database and if all the queries get executed successfully, it returns an array of update counts. We will use this method to insert/update the bulk of records.

**#5) Retrieve Results**

When we execute the queries using the executeQuery() method, the result will be stored in the ResultSet object. The returned ResultSet object will never be null even if there is no matching record in the table. ResultSet object is used to access the data retrieved from the Database.

ResultSet rs 1= statemnt1.executeQuery(QUERY));

We can use the executeQuery() method for the SELECT query. When someone tries to execute the insert/update query, it will throw SQLExecption with the message “**executeQuery method can not be used for update**”.

A ResultSet object points to the current row in the Resultset. To iterate the data in the ResultSet object, call the next() method in a while loop. If there is no more record to read, it will return FALSE.

ResultSet can also be used to update data in DB. We can get the data from ResultSet using getter methods such as getInt(), getString(), getDate(). We need to pass the column index or column name as the parameter to get the values using Getter methods.

We will get to know more about the ResultSet in the next tutorial.

**#6) Close Connection**

Finally, we are done with manipulating data in DB. Now we can close the JDBC connection. We need to make sure that we have closed the resource after we have used it. If we don’t close them properly we may end up out of connections.

When we close the connection object, Statement and ResultSet objects will be closed automatically.

conn.close();

From Java 7 onwards, we can close the JDBC connections automatically using a try-catch block. JDBC connection should be opened in the parenthesis of the try block. Inside the try block, you can do the database connections normally as we do.

Once the execution exits the try block, it will automatically close the connection. In this case, we don’t need to close the connection by calling conn.close method in the Java program.

|  |
| --- |
| **try**(Connection conn = DriverManager.getConnection(url, user, password))  {      //database connection and operation  } |

**Java JDBC Connection Example**

In this example, you will see how to implement the 6 basic steps to connect with database using JDBC in Java program.

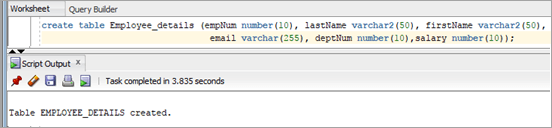
**Create Table**

Before that, first, create one table and add some entries into it.

**Below is the SQL query to create a table.**

|  |
| --- |
| **create** **table** employee\_details (empNum number(10), lastName **varchar**(50),   firstName **varchar**(50), email **varchar**(255) , deptNum number(10), salary number(10)); |

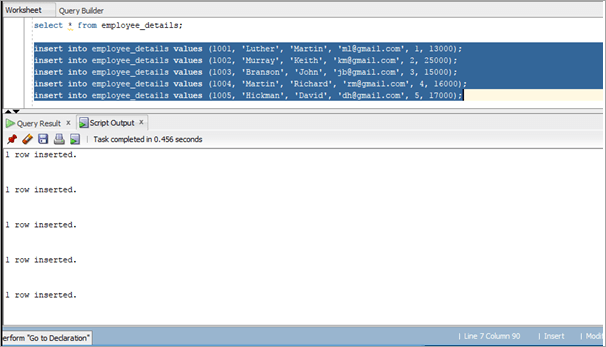
Created the “employee\_details” table in Oracle DB.

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2020/06/Create-employee_details.png)

**Insert Data Into Table**

Using the following queries, insert the data into the “employee\_details” table.

|  |
| --- |
| **insert** **into** employee\_details **values** (1001, 'Luther', 'Martin', 'ml@gmail.com', 1, 13000);  **insert** **into** employee\_details **values** (1002, 'Murray', 'Keith', 'km@gmail.com', 2, 25000);  **insert** **into** employee\_details **values** (1003, 'Branson', 'John', 'jb@gmail.com', 3, 15000);  **insert** **into** employee\_details **values** (1004, 'Martin', 'Richard', 'rm@gmail.com', 4, 16000);  **insert** **into** employee\_details **values** (1005, 'Hickman', 'David', 'dh@gmail.com', 5, 17000); |

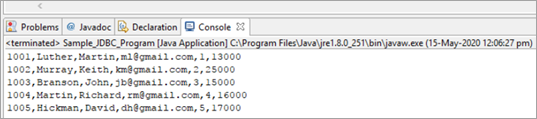
**[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2020/06/2020-06-08_10-50-08.png)**

**Java Program**

**Download the JDBC jar file and import it into the Java project.**

|  |
| --- |
| **package** com.STH.JDBC;      // import sql package to use it in our program  **import** java.sql.\*;    **public** **class** Sample\_JDBC\_Program {    **public** **static** **void** main(String[] args) **throws** ClassNotFoundException, SQLException {          // store the SQL statement in a string          String QUERY = "select \* from employee\_details";          //register the oracle driver with DriverManager          Class.forName("oracle.jdbc.driver.OracleDriver");          //Here we have used Java 8 so opening the connection in try statement  **try**(Connection conn = DriverManager.getConnection("jdbc:oracle:thin:system/pass123@localhost:1521:XE"))          {                  Statement statemnt1 = conn.createStatement();                  //Created statement and execute it                  ResultSet rs1 = statemnt1.executeQuery(QUERY);                  {                      //Get the values of the record using while loop  **while**(rs1.next())                      {  **int** empNum = rs1.getInt("empNum");                          String lastName = rs1.getString("lastName");                          String firstName = rs1.getString("firstName");                          String email = rs1.getString("email");                          String deptNum = rs1.getString("deptNum");                          String salary = rs1.getString("salary");                          //store the values which are retrieved using ResultSet and print it                      System.out.println(empNum + "," +lastName+ "," +firstName+ "," +email +","+deptNum +"," +salary);                      }                  }          }  **catch** (SQLException e) {              //If exception occurs catch it and exit the program              e.printStackTrace();          }         }      } |

**Output:**

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2020/06/2020-06-08_11-46-27.png)

**Key points to be noted:**

* First, we need to import the packages which we will be using in our Java program for the JDBC connection. So we can use the classes, subclasses, and interfaces in the packages.
* We need to register or load the driver with DriverManager before establishing a connection.
* After registering the driver, we can establish the connection and perform the operations.
* Using a statement interface we can create and execute the SQL query. For a simple SQL query, we can use the Statement interface. For insert/update/delete, we can use the PreparedStatement interface.
* After the statement execution, the results will be stored in the ResultSet object. We get the results from the ResultSet object using the next() method for more than 1 record.
* Once we are done with the database operation, we need to close the connection. So that the resource will be available for others to use.

**Frequently Asked Questions**

**Q #1) What are the fundamental steps to connect to DB in Java?**

**Answer:** There are 6 basic steps to connect to DB in Java.

**They are:**

1. Import-Package
2. Load driver
3. Establish Connection
4. Create and execute the statement
5. Retrieve results
6. Close Connection

# **JDBC Driver**

## Driver:

A driver is a software component that provides the facility to a computer to communicate with hardware.

## JDBC driver:

A driver is a software component that provides the facility to interact java application with the database.

## Types of JDBC drivers:

1. **JDBC-ODBC bridge driver.**
2. **Native-API driver.**
3. **Network-Protocol driver.**
4. **Thin driver.**

## 1. JDBC-ODBC bridge driver:

JDBC-ODBC bridge driver is a native code driver which uses ODBC driver to connect with the database. It converts JDBC method calls into ODBC function calls. It is also known as Type 1 driver.

## Advantages:

1. It can be used with any database for which an ODBC driver is installed.

## Disadvantages:

1. Performance is not good as it converts JDBC method calls into ODBC function calls.
2. ODBC driver needs to be installed on the client machine.
3. Platform dependent.

## 2. Native-API driver:

Native-API driver uses the client-side libraries of the database. It converts JDBC method calls into native calls of the database API. It is partially written in java. It is also known as Type 2 driver.

## Advantages:

1. It is faster than a JDBC-ODBC bridge driver.

## Disadvantages:

1. Platform dependent.
2. The vendor client library needs to be installed on the client machine.

## 3. Network-Protocol driver:

Network-Protocol driver is a pure java driver which uses a middle-tier to converts JDBC calls directly or indirectly into database specific calls. Multiple types of databases can be accessed at the same time. It is a platform independent driver. It is also known as Type 3 or MiddleWare driver.

## Advantages:

1. Platform independent.
2. Faster from Type1 and Type2 drivers.
3. It follows a three tier communication approach.
4. Multiple types of databases can be accessed at the same time.

## Disadvantages:

1. It requires database-specific coding to be done in the middle tier.

## 4. Thin driver:

Thin driver is a pure java driver which converts JDBC calls directly into the database specific calls. It is a platform independent driver. It is also known as Type 4 or Database-Protocol driver.

## Advantages:

1. Platform independent.
2. Faster than all other drivers.

## Disadvantages:

1. It is database dependent.
2. Multiple types of databases can’t be accessed at the same time.

# **Connect To MySql Database With JDBC Driver**

To connect with MySql database with JDBC driver follow the same basic steps discussed in previous tutorials. We have to know the following information to connect with MySql database:

**1. Driver class:** com.mysql.jdbc.Driver.

**2. Connection URL:**  
***Syntax:***

"jdbc:mysql://hostname:port/dbname","username", "password"

**Connection url for MySql database:** jdbc:mysql://localhost:3306/w3spoint

where 3306 is the port number and w3spoint is the database name.

**3. Username:** Username of MySql database, default is root.

**4. Password:** Password of MySql database.

## Example:

**JDBCMySqlTest.java**

|  |
| --- |
| **import** java.sql.Connection;  **import** java.sql.DriverManager;    ***/\*\****  ***\* This class is used to create a JDBC***  ***\* connection with MySql DB.***  ***\* @author w3spoint***  ***\*/***  **public** **class** JDBCOracleTest {  *//JDBC and database properties.*  **private** **static** **final** String DB\_DRIVER =  "com.mysql.jdbc.Driver";  **private** **static** **final** String DB\_URL =  "jdbc:mysql://localhost:3306/w3spoint";  **private** **static** **final** String DB\_USERNAME = "root";  **private** **static** **final** String DB\_PASSWORD = "root";    **public** **static** **void** main(String args[]){  Connection conn = **null**;  **try**{  *//Register the JDBC driver*  **Class**.forName(DB\_DRIVER);    *//Open the connection*  conn = DriverManager.  getConnection(DB\_URL, DB\_USERNAME, DB\_PASSWORD);    **if**(conn != **null**){  System.out.println("Successfully connected.");  }**else**{  System.out.println("Failed to connect.");  }  }**catch**(Exception e){  e.printStackTrace();  }  }  } |

## Output:

|  |
| --- |
| Successfully connected. |

**Step by Step JDBC Program Example**

Here are the different steps to connect and working with the database through Java code.

**Steps to develop JDBC Program :**

To implement the JDBC code in Java program, typically we have 6 different steps, are listed below.

* Load a JDBC Driver class
* Establish a Connection
* Create a Statement
* Execute Sql queries
* Process the ResultSet
* Close Connection

To write a simple **JDBC program**, we should follow these 6 different steps. Lets understand indetail for each step.

**JDBC Program Step 1 :**

**Load a JDBC Driver :** To load a class at runtime into JVM, we need to call a static method called **forName()** of java.lang.Class. By calling the forName() method we can load the JDBC Driver class into the JVM.

class.forName("sun.jdbc.odbc.JdbcOdbcDriver);

**JDBC Program Step 2 :**

**Establish a Connection :**By using the DriverManager class, we can establish the connection to the database. By calling the **getConnection(String url, String user,String password)** static factory method in DriverManager class, we can obtain a Connection object. To get the connection object we need to proved the connection url as a first parameter and database username and password as second and third parameters. Like below.

Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/onlinetutorialspoint","root","123456");

**JDBC Program Step 3 :**

**Create a Statement :**In order to send the SQL commands to database from our java program, we need Statement object. We can get the Statement object by calling the **createStatement()**method on **connection.**

Statement stmt = con.createStatement();

**JDBC Program Step 4 :**

**Execute Sql queries :**Inorder to execute the SQL commands on database, Statement interface provides provides three different methods:

* executeUpdate()
* executeQuery()
* execute()

When we want to run the **non-select** operations then we  can choose **executeUpdate()**

int count = stmt.executeUpdate("non-select command");

When we want to execute **select** operations then we  can choose **executeQuery()**

ResultSet rs = stmt.executeQuery("select command");

When we want to run both select and non-select operations, then we can use **execute()**

boolean isTrue = stmt.executeQuery("select / non-select command");

**JDBC Program Step 5 :**

**Process the ResultSet :** For the select operations, we use **executeQuery()** method. The executed query returns the data in the form of ResultSet object. To process the data we need to go through the ResultSet.

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

while(rs.next()){

System.out.println(rs.getInt(1));

}

**JDBC Program Step 6 :**

**Close Connection :** Last but not least, finally we need to close the connection object. It is very important step to close the connection. Else we may get [*JDBCConnectionException*](https://docs.jboss.org/hibernate/orm/3.2/api/org/hibernate/exception/JDBCConnectionException.html) exception.

con.close();

**Complete JDBC Program Example :**

Jdbc\_Select\_Example.java

package com.onlinetutorialspoint.jdbc;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.Statement;

public class Jdbc\_Select\_Example {

public static void main(String[] args) throws Exception {

// Step - I

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

// Step - II

Connection con = DriverManager.getConnection(

"jdbc:mysql://localhost:3306/jdbc", "root", "123456");

// Step -III

Statement stmt = con.createStatement();

// Step - IV

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

// Step - V

while (rs.next()) {

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

+ rs.getString(3));

}

// Step - VI

rs.close();

stmt.close();

con.close();

}

}