JDBC

(Java Database Connectivity)

JDBC or Java Database Connectivity is a Java API to connect and execute the query with the database. It is a specification from Sun microsystems that provides a standard abstraction (API or Protocol) for java applications to communicate with various databases. It provides the language with java database connectivity standards. It is used to write programs required to access databases. JDBC, along with the database driver, can access databases and spreadsheets. The enterprise data stored in a relational database (RDB) can be accessed with the help of JDBC APIs.

Definition of JDBC (Java Database Connectivity)

JDBC is an API (Application programming interface) used in java programming to interact with databases. *The classes and interfaces of JDBC* allow the *application to send* requests *made by users to the specified database.*

Purpose of JDBC

Enterprise applications created using the JAVA EE technology need to interact with databases to store application-specific information. So, interacting with a database requires efficient database connectivity, which can be achieved by using that database driver. This drivers respective database's are used with JDBC to interact or communicate with various kinds of databases such as Oracle, MS Access, Mysql, and SQL server database.

Components of JDBC

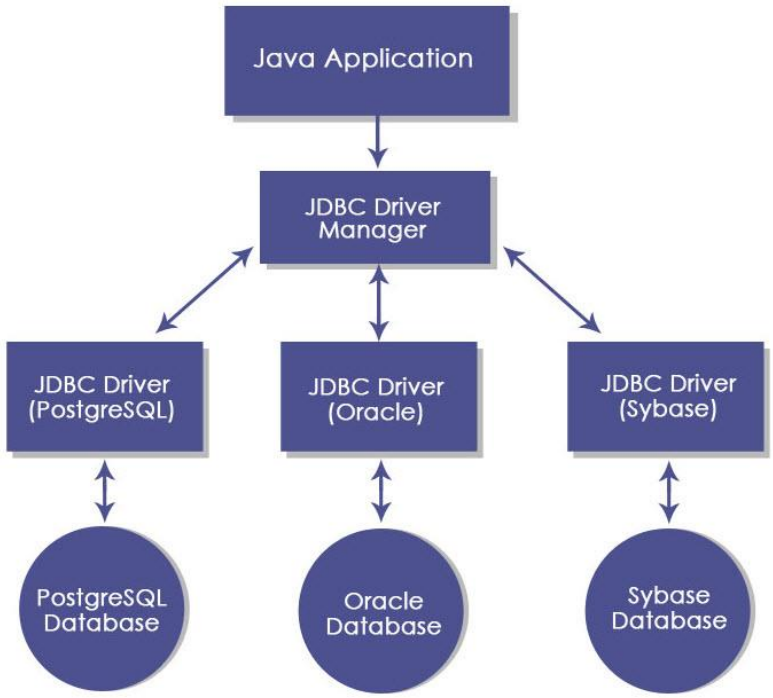
There are generally four main components of JDBC through which it can interact with a database. They are as mentioned below:

1. JDBC API: It provides various methods and interfaces for easy communication with the database. It provides two packages as follows, which contain the java SE and Java EE platforms to exhibit WORA (write once run everywhere) capabilities.
java.sql.*;

It also provides a standard to connect a database to a client application.

3. JDBC DRIVER MANAGER: It loads a database-specific driver in an application to establish a connection with a database. It is used to make a database-specific call to the database to process the user request.
4. JDBC Test suite: It is used to test the operation (such as insertion, deletion, updation) being performed by JDBC Drivers.
5. RESULTSET: A table of data representing a database result set, which is usually generated by executing a statement that queries the database.

Architecture of JDBC



Architecture of JDBC

Description:

1. **Application:** It is a java applet or a servlet that communicates with a data source.
2. **The JDBC API:** The JDBC API allows Java programs to execute SQL statements and retrieve results. Some of the important classes and interfaces defined in JDBC API are as follows:
3. **DriverManager:** It plays an important role in the JDBC architecture. It uses some database-specific drivers to effectively connect enterprise applications to databases.
4. **JDBC drivers:** To communicate with a data source through JDBC, you need a JDBC driver that intelligently communicates with the respective data source.

Interfaces of JDBC API

A list of popular *interfaces* of JDBC API is given below:

- Driver interface
- Connection interface
- Statement interface
- PreparedStatement interface
- CallableStatement interface
- ResultSet interface
- ResultSetMetaData interface
- DatabaseMetaData interface
- RowSet interface

Classes of JDBC API

A list of popular *classes* of JDBC API is given below:

- DriverManager class
- Blob class
- Clob class
- Types class

Working of JDBC

Java application that needs to communicate with the database has to be programmed using JDBC API. JDBC Driver supporting data sources such as Oracle and SQL server has to be added in java application for JDBC support which can be done dynamically at run time. This JDBC driver intelligently communicates the respective data source.

STEPS TO CONNECT TO DATABASE :

There are 5 steps to connect any java application with the database using JDBC. These steps are as follows:

- Import the packages
- Load/Register the Driver class
- Establish connection
- Create statement
- Execute statement
- Process the results
- Close connection

Firstly we have to download the respective driver and add it to the project libraries.

I. Register the Driver Class

Two ways :

- The **forName()** method of Class class is used to register the driver class. This method is used to dynamically load the driver class.
  EX : for connecting to mysql database.

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

- By using Driver object.
  EX :

```
Driver d = new Driver();

DriverManager.registerDriver(d);
```

II. ESTABLISH CONNECTION :

- We can establish the connection by using getConnection() of DriverManager class in java.sql package.
- The return type is Connection Object.
- getConnection() is an overloaded method.
- There are three ways to establish connection.
  i. getConnection(url,username,password)
  ii. getConnection(url)
  iii. getConnection(url,properties)

i. **getConnection(url,username,password)**

```
String url="jdbc:mysql://localhost:3306/javabatch";
String userName="root";
String password="root";

Connection con = DriverManager.getConnection(url, userName,
password);
```

ii. **getConnection(url)**

```
String url =
"jdbc:mysql://localhost:3307/javabatch?user=root&password=root";

Connection con=DriverManager.getConnection(url);
```

iii. **getConnection(url , properties)**

first create a properties file with user and password.
(EX: mydbinfo.properties)

```
Properties properties = new Properties();
InputStream inputStream = new FileInputStream("mydbinfo.properties");
properties.load(inputStream);
Connection con = DriverManager.getConnection(url,properties);
```

III. Create Statement :

Two ways :

a) Statement
b) PreparedStatement

a. Statement

```
Statement stm= con.createStatement();
```

b. PreparedStatement

You have to pass the query here ,

```
PreparedStatement p=con.prepareStatement(query);
```

IV. Execute Queries :

Using Statement : Here we pass query in execute statement.
Used for static queries.

a) `stm.execute(sql); (boolean)`

b) `stm.executeUpdate(sql);(int)`

c) `stm.executeQuery(sql);(resultList obj)`

Using PreparedStatement : Here we pass query while creating preparedStatement object only.
It can be used for dynamic queries.

a) `preparedstm.execute(); (boolean)`

b) `preparedstm.executeUpdate();(int)`

c) `preparedstm.executeQuery();(resultList obj)`

V. Close Connection :

Each machine has a limited number of connections (separate thread)

- If connections are not closed the system will run out of resources and freeze

```
connection.close();
```

-Correct way of closing connection is always by using finally block.

RESULTSET INTERFACE :

A table of data representing a database result set, which is usually generated by executing a statement that queries the database.

A ResultSet object maintains a cursor pointing to its current row of data. Initially the cursor is positioned before the first row. The next() method moves the cursor to the next row, and because it returns false when there are no more rows in the ResultSet object, it can be used in a while loop to iterate through the result set.

A default ResultSet object is not updatable and has a cursor that moves forward only. Thus, you can iterate through it only once and only from the first row to the last row. It is possible to produce ResultSet objects that are scrollable and/or updatable. The following code fragment, in which con is a valid Connection object, illustrates how to make a result set that is scrollable and insensitive to updates by others, and that is updatable. See ResultSet fields for other options.

EX :

```
Statement stmt = con.createStatement();

ResultSet rs = stmt.executeQuery("SELECT a, b FROM TABLE2");
```

SOME IMPORTANT METHODS OF RESULTSET :

1. public boolean next(): is used to move the cursor to the one row next from the current position.
2. **public boolean previous():** is used to move the cursor to the one row previous from the current position.
3. **public int getInt(int columnIndex):** is used to return the data of specified column index of the current row as int.
4. **public int getInt(String columnName):** is used to return the data of specified column name of the current row as int.
5. **public String getString(int columnIndex):** is used to return the data of specified column index of the current row as String.
6. **public String getString(String columnName):** is used to return the data of specified column name of the current row as String.

   NOTE :

   Like this , we have overloaded getter and setter methods for all primitive types.

   DIFFERENCE BETWEEN EXCECUTE() , EXECUTEUPDATE() AND EXECUTEQUERY :

- **executeQuery() Vs executeUpdate() Vs execute()** are the methods of **java.sql.Statement** interface of JDBC which are used to execute SQL statements.
- **executeQuery()** command used for getting the data from database whereas **executeUpdate()** command used for insert,update,delete or **execute()** command used forany kind of operations.

Comparison Chart

executeQuery()	executeUpdate()	execute()
executeQuery() method used to retrieve some data from database.	**executeUpdate()** method used for update or modify database.	**execute()** use for any SQL statements.
It returns an object of the class ResultSet executeQuery (String sql) throws SQLException	It returns an integer value. int executeUpdate(String sql) throws SQLException	It returns a boolean value. int executeUpdate(String sql) throws SQLException
This method is normally used to **execute SELECT** queries.	This method Is used to **execute non SELECT** queries. • **DML** as INSERT, DELETE, UPDATE or • **DDL** as CREATE. DROP	This method can be used to execute **any type** of SQL statement.
Example: • ResultSet Ts= stmt.executeQuery(query);	Example: • int i= stmt.executeUpdate(query);	Example: • Boolean b= stmt.execute(query);

BATCH EXECUTION :

Batch Processing allows you to group related SQL statements into a batch and submit them with one call to the database.

When you send several SQL statements to the database at once, you reduce the amount of communication overhead, thereby improving performance.

- The **addBatch()** methodof *Statement,PreparedStatement,* and *CallableStatement* is used to add individual statements to the batch. The **executeBatch()** is used to start the execution of all the statements grouped together.
- The **executeBatch()** returns an array of integers, and each element of the array represents the update count for the respective update statement.
- Just as you can add statements to a batch for processing, you can remove them with the **clearBatch()** method. This method removes all the statements you added with the addBatch() method. However, you cannot selectively choose which statement to remove.

EX :

```
package batchexecutions;

import property.Employee;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.SQLException;

import java.util.List;


import helper.HelperClass;


public class EmployeeBatchExecution {


    HelperClass helperClass=new HelperClass();

    Connection connection=helperClass.getConnection();
```

```
public void executeEmployee(List<Employee> list) {
        String query="INSERT INTO employee VALUES(?,?,?)";

        try {
            PreparedStatement
preparedStatement=connection.prepareStatement(query);
            for(Employee employee: list) {
                preparedStatement.setInt(1,
employee.getId());
                preparedStatement.setString(2,
employee.getName());
                preparedStatement.setString(3,
employee.getEmail());

                preparedStatement.addBatch();
                System.out.println("employee with ID :
"+employee.getId()+"is added to batch");
            }
            preparedStatement.executeBatch();
            System.out.println("All Employees are saved");
        } catch (SQLException e) {

            e.printStackTrace();
        }
        finally {
            try {
                connection.close();
            } catch (SQLException e) {
                // TODO Auto-generated catch block
```

```
                    e.printStackTrace();
                }
            }

        }
}
```

DELIMETERS OR PLACEHOLDERS(?) :

1. The delimiters or placeholders are used in sql query.
2. We can assign the values for placeholders using statement and preparedStatement interface set methods.
3. If there are more than one place holder we give numbering to each one. EX :

```
public void saveStudent(int id, String name, String email) {
        try {

                Connection con=help.getConnectionObj();
                String sql="INSERT INTO student VALUES(?,?,?)";

                PreparedStatement p= con.prepareStatement(sql);

                p.setInt(1, id);
                p.setString(2, name);
                p.setString(3, email);

                System.out.println("Data Inserted");

        }  catch (SQLException e) {

                e.printStackTrace();}}
```