Top of Form

**JDBC ARCHITECTURE**

Java application calls the JDBC library. JDBC loads a driver which talks to the database.

JDBC contains three components: Application, Driver Manager, Driver.

The user application invokes JDBC methods to send SQL statements to the database and retrieves results. JDBC driver manager is used to connect Java applications to the correct JDBC driver . JDBC driver test suite is used to ensure that the installed JDBC driver is JDBC Compliant.

**JDBC ARCHITECTURE/JDBC Driver Types**

•**Type 1**

• JDBC-ODBC Bridge

•**Type 2**

•Native API, partially java driver

•**Type 3**

•JDBC Network Driver, partially java

•**Type 4**

•Native-protocol pure Java driver (100% Java )

**Which Driver should be Used?**

If you are accessing one type of database, such as Oracle, Sybase, or IBM, the preferred driver type is 4.

If your Java application is accessing multiple types of databases at the same time, type 3 is the preferred driver.

Type 2 drivers are useful in situations, where a type 3 or type 4 driver is not available yet for your database.

The type 1 driver is not considered a deployment-level driver, and is typically used for development and testing purposes only.

#### 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 class

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

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

#### 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");

#### 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

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));

}

#### 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

con.close();

###### Note: Since Java 7, JDBC has ability to use try-with-resources statement to automatically close resources of type Connection, ResultSet, and Statement.

It avoids explicit connection closing step.

#### Java Database Connectivity with Oracle

To connect java application with the oracle database, we need to follow 5 following steps.

In this example, we are using Oracle 10g as the database. So we need to know following information for the oracle database:

**Driver class:** The driver class for the oracle database is oracle.jdbc.driver.OracleDriver.

**Connection URL:** The connection URL for the oracle10G database is jdbc:oracle:thin:@localhost:1521:xe where jdbc is the API, oracle is the database, thin is the driver, localhost is the server name on which oracle is running, we may also use IP address, 1521 is the port number and XE is the Oracle service name. You may get all these information from the **tnsnames.ora** file.

**Username:** The default username for the oracle database is system.

**Password:**It is the password given by the user at the time of installing the oracle database.

Create a TableBefore establishing connection, let's first create a table in oracle database. Following is the SQL query to create a table.

1. create table emp(id number(10),name varchar2(40),age number(3));

#### Example to Connect Java Application with Oracle database

In this example, we are connecting to an Oracle database and getting data from **emp** table. Here, **system** and **oracle** are the username and password of the Oracle database.

1. **import** java.sql.\*;
2. **class** OracleCon{
3. **public** **static** **void** main(String args[]){
4. **try**{
5. //step1 load the driver class
6. Class.forName("oracle.jdbc.driver.OracleDriver");
8. //step2 create  the connection object
9. Connection con=DriverManager.getConnection(
10. "jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
12. //step3 create the statement object
13. Statement stmt=con.createStatement();
15. //step4 execute query
16. ResultSet rs=stmt.executeQuery("select \* from emp");
17. **while**(rs.next())
18. System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
20. //step5 close the connection object
21. con.close();
23. }**catch**(Exception e){ System.out.println(e);}
25. }
26. }

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

To connect java application with the Oracle database ojdbc14.jar file is required to be loaded.

**Link::** [**https://static.javatpoint.com/src/jdbc/ojdbc14.jar**](https://static.javatpoint.com/src/jdbc/ojdbc14.jar)

#### Two ways to load the jar file:

1. paste the ojdbc14.jar file in jre/lib/ext folder
2. set classpath

#### 1) paste the ojdbc14.jar file in JRE/lib/ext folder:

Firstly, search the ojdbc14.jar file then 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:

Firstly, search the ojdbc14.jar file then open command prompt and write:

1. C:>set classpath=c:\folder\ojdbc14.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 ojdbc14.jar by appending ojdbc14.jar;.; as C:\oraclexe\app\oracle\product\10.2.0\server\jdbc\lib\ojdbc14.jar;.;

#### Statement interface

The **Statement interface** provides methods to execute queries with the database. The statement interface is a factory of ResultSet i.e. it provides factory method to get the object of ResultSet.

#### Commonly used methods of Statement interface:

The important methods of Statement interface are as follows:

1) public ResultSet executeQuery(String sql): is used to execute SELECT query. It returns the object of ResultSet.

2) public int executeUpdate(String sql): is used to execute specified query, it may be create, drop, insert, update, delete etc.

3) public boolean execute(String sql): is used to execute queries that may return multiple results.

4) public int[] executeBatch(): is used to execute batch of commands.

#### Example of Statement interface

Let’s see the simple example of Statement interface to insert, update and delete the record.

1. **import** java.sql.\*;
2. **class** FetchRecord{
3. **public** **static** **void** main(String args[])**throws** Exception{
4. Class.forName("oracle.jdbc.driver.OracleDriver");
5. Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
6. Statement stmt=con.createStatement();
8. //stmt.executeUpdate("insert into emp765 values(33,'Irfan',50000)");
9. //int result=stmt.executeUpdate("update emp765 set name='Vimal',salary=10000 where id=33");
10. **int** result=stmt.executeUpdate("delete from emp765 where id=33");
11. System.out.println(result+" records affected");
12. con.close();
13. }}

#### ****PreparedStatement interface****

The PreparedStatement interface is a subinterface of Statement. It is used to execute parameterized query.

Let's see the example of parameterized query:

1. String sql="insert into emp values(?,?,?)";

As you can see, we are passing parameter (?) for the values. Its value will be set by calling the setter methods of PreparedStatement.

#### ****Why use PreparedStatement?****

**Improves performance**: The performance of the application will be faster if you use PreparedStatement interface because query is compiled only once.

#### ****How to get the instance of PreparedStatement?****

The prepareStatement() method of Connection interface is used to return the object of PreparedStatement. Syntax:

1. **public** PreparedStatement prepareStatement(String query)**throws** SQLException{}

#### ****Methods of PreparedStatement interface****

The important methods of PreparedStatement interface are given below:

#### ****Example of PreparedStatement interface that inserts the record****

First of all create table as given below:

1. create table emp(id number(10),name varchar2(50));

Now insert records in this table by the code given below:

1. **import** java.sql.\*;
2. **class** InsertPrepared{
3. **public** **static** **void** main(String args[]){
4. **try**{
5. Class.forName("oracle.jdbc.driver.OracleDriver");
7. Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
9. PreparedStatement stmt=con.prepareStatement("insert into Emp values(?,?)");
10. stmt.setInt(1,101);//1 specifies the first parameter in the query
11. stmt.setString(2,"Ratan");
13. **int** i=stmt.executeUpdate();
14. System.out.println(i+" records inserted");
16. con.close();
18. }**catch**(Exception e){ System.out.println(e);}
20. }
21. }

#### ****Example of PreparedStatement interface that updates the record****

1. PreparedStatement stmt=con.prepareStatement("update emp set name=? where id=?");
2. stmt.setString(1,"Sonoo");//1 specifies the first parameter in the query i.e. name
3. stmt.setInt(2,101);
5. **int** i=stmt.executeUpdate();
6. System.out.println(i+" records updated");

#### ****Example of PreparedStatement interface that deletes the record****

1. PreparedStatement stmt=con.prepareStatement("delete from emp where id=?");
2. stmt.setInt(1,101);
3. **int** i=stmt.executeUpdate();
4. System.out.println(i+" records deleted");

#### ****Example of PreparedStatement interface that retrieve the records of a table****

1. PreparedStatement stmt=con.prepareStatement("select \* from emp");
2. ResultSet rs=stmt.executeQuery();
3. **while**(rs.next()){
4. System.out.println(rs.getInt(1)+" "+rs.getString(2));
5. }

#### ****Example of PreparedStatement to insert records until user press n****

1. **import** java.sql.\*;
2. **import** java.io.\*;
3. **class** RS{
4. **public** **static** **void** main(String args[])**throws** Exception{
5. Class.forName("oracle.jdbc.driver.OracleDriver");
6. Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
8. PreparedStatement ps=con.prepareStatement("insert into emp130 values(?,?,?)");
10. BufferedReader br=**new** BufferedReader(**new** InputStreamReader(System.in));
12. **do**{
13. System.out.println("enter id:");
14. **int** id=Integer.parseInt(br.readLine());
15. System.out.println("enter name:");
16. String name=br.readLine();
17. System.out.println("enter salary:");
18. **float** salary=Float.parseFloat(br.readLine());
20. ps.setInt(1,id);
21. ps.setString(2,name);
22. ps.setFloat(3,salary);
23. **int** i=ps.executeUpdate();
24. System.out.println(i+" records affected");
26. System.out.println("Do you want to continue: y/n");
27. String s=br.readLine();
28. **if**(s.startsWith("n")){
29. **break**;
30. }
31. }**while**(**true**);
33. con.close();
34. }}

#### ****Java CallableStatement Interface****

CallableStatement interface is used to call the **stored procedures and functions**.

We can have business logic on the database by the use of stored procedures and functions that will make the performance better because these are precompiled.

Suppose you need the get the age of the employee based on the date of birth, you may create a function that receives date as the input and returns age of the employee as the output.

#### ****What is the difference between stored procedures and functions.****

The differences between stored procedures and functions are given below:

#### ****How to get the instance of CallableStatement?****

The prepareCall() method of Connection interface returns the instance of CallableStatement. Syntax is given below:

1. **public** CallableStatement prepareCall("{ call procedurename(?,?...?)}");

The example to get the instance of CallableStatement is given below:

CallableStatement stmt=con.prepareCall("{call myprocedure(?,?)}");

It calls the procedure myprocedure that receives 2 arguments.

#### ****Full example to call the stored procedure using JDBC****

To call the stored procedure, you need to create it in the database. Here, we are assuming that stored procedure looks like this.

1. create or replace procedure "INSERTR"
2. (id IN NUMBER,
3. name IN VARCHAR2)
4. is
5. begin
6. insert into user12 values(id,name);
7. end;
8. /

The table structure is given below:

create table user12(id number(10), name varchar2(200));

In this example, we are going to call the stored procedure INSERTR that receives id and name as the parameter and inserts it into the table user12. Note that you need to create the user12 table as well to run this application.

1. **import** java.sql.\*;
2. **public** **class** Proc {
3. **public** **static** **void** main(String[] args) **throws** Exception{
5. Class.forName("oracle.jdbc.driver.OracleDriver");
6. Connection con=DriverManager.getConnection( "jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
8. CallableStatement stmt=con.prepareCall("{call insertR(?,?)}");
9. stmt.setInt(1,1011);
10. stmt.setString(2,"Amit");
11. stmt.execute();
13. System.out.println("success");
14. }
15. }

Now check the table in the database, value is inserted in the user12 table.

Bottom of Form

Top of Form

**Reference Links:**

https://www.tutorialspoint.com/jdbc/jdbc-statements.htm

https://docs.oracle.com/javase/tutorial/jdbc/basics/processingsqlstatements.html

https://docs.oracle.com/javase/7/docs/api/java/sql/Statement.html

https://www.javatpoint.com/Statement-interface

https://www.javatpoint.com/PreparedStatement-interface

https://www.javatpoint.com/CallableStatement-interface

**Video Link:**

https://youtu.be/eEqPrlu28Sc

https://youtu.be/v5vLuCBv8vg

Bottom of Form