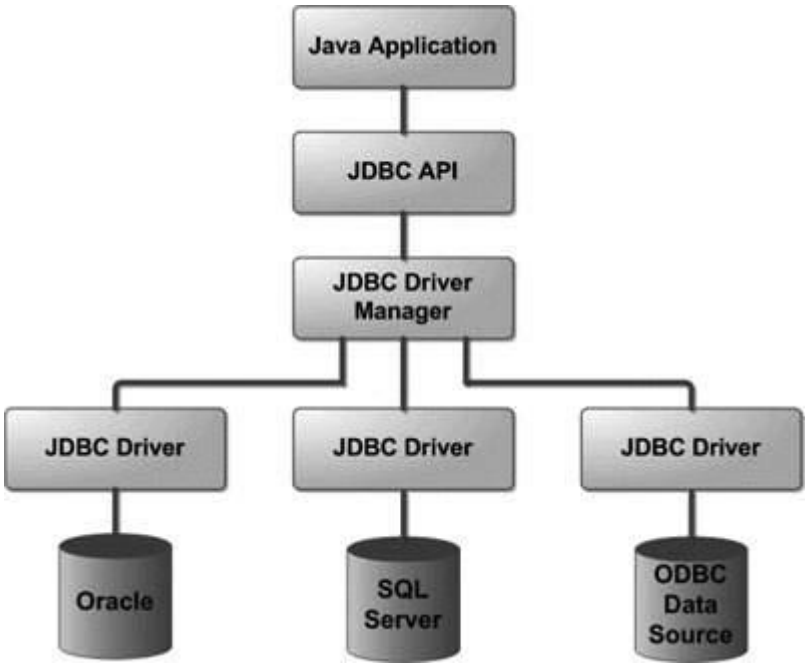
Chap1: JDBC

Introduction:

- **JDBC** stands for **Java Database Connectivity.**
- **JDBC** 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.

Architecture of JDBC



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.
3. **DriverManager:** Acts as an interface between a java application and a database. It contains drivers. Driver sends the request of a java application to the database. After processing the request, the database sends the response back to the driver.

4. **JDBC drivers:** To communicate with a data source through JDBC, you need a JDBC driver that intelligently communicates with the respective data source.

Components of JDBC

JDBC has four major components that are used for the interaction with the database.

1. JDBC API
2. JDBC Test Suite
3. JDBC Driver Manger
4. JDBC ODBC Bridge Driver

1) JDBC API: JDBC API provides various interfaces and methods to establish easy connection with different databases.

1. javax.sql.*;
2. java.sql.*;

2) JDBC Test suite:

JDBC Test suite facilitates the programmer to test the various operations such as deletion, updation, insertion that are being executed by the JDBC Drivers.

3) JDBC Driver manager:

- JDBC Driver manager loads the database-specific driver into an application in order to establish the connection with the database.
- The JDBC Driver manager is also used to make the database-specific call to the database in order to do the processing of a user request.

4) JDBC-ODBC Bridge Drivers:

- JDBC-ODBC Bridge Drivers are used to connect the database drivers to the database.
- The bridge does the translation of the JDBC method calls into the ODBC method call.
- It makes the usage of the sun.jdbc.odbc package that encompasses the native library in order to access the ODBC (Open Database Connectivity) characteristics.

JDBC API

- Interfaces
- Classes
- url
- package

1)Interfaces

- **Driver interface:** Represents a database driver. All JDBC driver classes must implement the Driver interface
- **Connection :** enables you to establish a connection between a java application & a database.
- **Statement :** enables you to execute SQL statements.
- **PreparedStatement:** allows to execute dynamic SQL statements and stored procedures
- **CallableStatement:** provides methods for executing stored procedures that return OUT parameter values
- **ResultSet :** Represent the information retrieved from a database.
- **DatabaseMetaData:** enables to get information about database
- **ResultSetMetaData:** allows to get information about a returned result

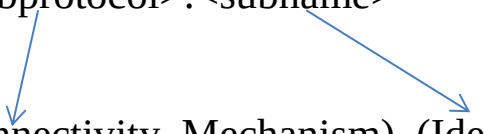
2)Classes

- **SQLException :** Provides information about the exception that occur while interacting with database.

- **DriverManager :** responsible for loading Jdbc drivers and creating connection objects

3)URL

- Provides a way of identifying a database so that the appropriate driver will recognize it and establish a connection with it
- Jdbc:<subprotocol>:<subname>



(Database Connectivity Mechanism) (Identify database)

Steps to Create JDBC Application:

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** ClassNotFound Exception

Example

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

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

2) Create the connection object

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

Syntax of getConnection() method

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

Example

Connection con=DriverManager.getConnection(" jdbc:odbc:MYDSN");

---

3) Create the Statement object

- The createStatement() method of Connection interface is used to create state
- 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

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

---

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();
```

Types of statements:

There are three types of statements in JDBC namely, Statement, Prepared Statement, Callable statement.

1)Statement

- The Statement interface represents the static SQL statement.
- It helps you to create a general purpose SQL statements using Java.

Creating a statement

- You can create an object of this interface using the **createStatement()** method of the **Connection** interface.

- Create a statement by invoking the **createStatement()** method as shown below.

```
Statement stmt = null;
stmt = conn.createStatement( );
```

Executing the Statement object

Once you have created the statement object you can execute it using one of the execute methods namely, execute(), executeUpdate() and, executeQuery().

ResultSet executeQuery(<SQL Statement>):

- It executes SQL statement.
- This method is used when you need to retrieve data from a database table using the SELECT statement.

Statement stmt = cnn.createStatement();

ResultSet rs = stmt.executeQuery(<SQL statement>)

int executeUpdate(<SQL statement>):

- execute the SQL statement and returns the number of data rows that are affected after processing the SQL statement.
- When you modify data in a database table using INSERT, DELETE, UPDATE.

Statement stmt = cnn.createStatement();

int count = stmt.executeUpdate(<SQL Statement>);

Boolean execute((<SQL statement>): execute SQL statement & returns a Boolean value.

2)CallableStatement

- Callable statements are implemented by the CallableStatement object.

- A CallableStatement is a way to **execute stored procedures** in a JDBC-compatible database.

Creating a CallableStatement

You can create an object of the **CallableStatement** (interface) using

the **prepareCall()** method of the **Connection** interface.

This method accepts a string variable representing a query to

call the stored procedure and returns a **CallableStatement** object.

A CallableStatement can have input parameters or, output parameters or, both.

To pass input parameters to the procedure call you can use place holder and set values to these using the setter methods (setInt(), setString(), setFloat()) provided by the CallableStatement interface.

Suppose, you have a procedure name myProcedure in the database you can prepare a callable statement as:

```
//Preparing a CallableStatement
CallableStatement cstmt = con.prepareCall("{call myProcedure(?, ?, ?)}");
```

Setting values to the input parameters

You can set values to the input parameters of the procedure call using the setter methods.

These accept two arguments one is an integer value representing the placement index of the input parameter and the other is an int or, String or, float etc… representing the value you need to pass an input parameter to the procedure.

Note: Instead of index you can also pass the name of the parameter in String format.

```
cstmt.setString(1, "Raghav");
cstmt.setInt(2, 3000);
cstmt.setString(3, "Hyderabad");
```

Executing the Callable Statement

Once you have created the CallableStatement object you can execute it using one of the **execute()** method.

```
cstmt.execute();
```

3)PreparedStatement

- A PreparedStatement, in contrast to a CallableStatement, is used for SQL statements that are executed multiple times with different values.
- For instance, you might want to insert several values into a table, one after another.
- The advantage of the PreparedStatement is that it is pre-compiled, reducing the overhead of parsing SQL statements on every execution.
- Initially, this statement uses place holders "**?**" instead of parameters, later on, you can pass arguments to these dynamically using the **setXXX()** methods of the **PreparedStatement** interface.
- e.g.
  String qry ="select Books.Price, Books.Title" +"from Books, Publishers" +"where Books.Pub_Id = Publishers.Pub_Id"+ "and Publishers.Name = ?";

  PreparedStatement pst = con.preparedStatement(qry);

Before executing the Prepared Statement, we must bind the host variable to actual values with set method.

e.g.

We can set a string to publisher name as follows.

pst.setString(1, publisher);

where,

first argument is position of host variable.

second argument is actual value for host variable.

All the host variable stay bound unless we change it with set method.

Then we can execute prepared query as follows.

ResultSet rs = pst.executeQuery();

Working with ResultSet

ResultSet interface

The object of ResultSet maintains a cursor pointing to a row of a table. Initially, cursor points to before the first row.

next(): Moves the current row in Result Set by one.

getXXX(int colnum): Returns value of the specified column for current row.

getXXX(String cname): Returns value of the specified column for current row.

Where,

XXX is a type such as Int, String, Double, Date, etc.

close(): closes the current result set.

Scrollable ResultSet:

It allows user to move both direction, and also jump to any specific position.

By default Resultset is not scrollable. We can obtain the scrollable result set as follows.

For Statement:

Statement st = con.createStatement(type, concurrency);

For Prepared Statement:

PreparedStatement pst = con.preparedStatement(cmd, type, concurrency)

The possible values of the type and concurrency are as follows.

For type:

TYPE_FORWARD_ONLY: Resultset is not scrollable **(Default)**.

TYPE_SCROLL_INSENSITIVE: Resultset is scrollable but not sensitive to database changes.

TYPE_SCROLL_SENSITIVE: Resultset is scrollable and sensitive to database changes.

For concurrency:

CONCUR_READ_ONLY: Resultset can not be used to update the database. **(Default)**.

CONCUR_UPDATABLE: Resultset can be used to update the database.

Result can be scrolled using following method.

previous(): It is used to scroll backward. Returns true if success.

next(): It is used to scroll forward. Returns true if success.

relative(int n): It moves cursor n records either forward or backward.If n is positive, it moves forward.If n is negative, it moves backward.If n is zero, it has no effect.

absolute(int n): It moves the cursor to specific row number i.e. n.

getRow(): It returns the current row number where cursor is located.

Updatable ResultSet:

It allows user to update the entries through programming automatically.This type of Resultset is not scrollable.

But usually user want scrollable Resultset while editing or updating.

We can obtain the updatable Resultset as follows:

```
Statement st = con.createStatement(
ResultSet.TYPE_SCROLL_INSESNSITIVE,
ResultSet.CONCUR_UPDATABLE);
```

e.g.

```
String qry = "select * from Books";

ResultSet rs = st.executeQuery(qry);

while(rs.next())

{

double prc = rs.getDouble("Price");

rs.updateDouble("Price", prc + 300);

rs.updateRow();

}
```

There are several methods are used to update row value for specific column.

updateXXX(int col_number):

updateXXX(String col_name):

It changes the only row value for specific column.

where

XXX :- specify Int, Double, String, Date, etc.

updateRow(): It sends all updates in the current row to database.