**UNIT - 5**

**DataBase Connectivity using JDBC driver interface**

**5.1 know about JDBC**

JDBC Standards for Java DataBase Connectivity. It is Standard API(Application Programming Interface) for all Java programs to connect to wide variety of databases. The JDBC API is available in 2 packages.

* Core API **java.sql.**
* Standard extension to JDBC API **javax.sql**(supports connection pooling, transactions, etc.)

Before JDBC, ODBC API was the database API to connect and execute the query with the database. But, ODBC API uses ODBC driver which is written in C language (i.e. platform dependent and unsecured). That is why Java has defined its own API (JDBC API) that uses JDBC drivers (written in Java language).

**There are few steps to connect to a database and retrieve/insert/update the database:**

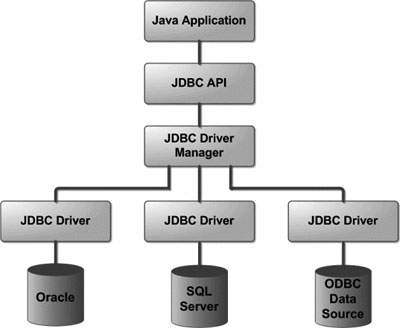
* Load the driver
* Establish the connection
* Create the statements
* Execute the query
* Obtain results
* Close connection

**5.2 Understand JDBC Architecture:**

The JDBC API supports both two-tier and three-tier processing models for database access but in general, JDBC Architecture consists of two layers −

* **JDBC API:** This provides the application-to-JDBC Manager connection.
* **JDBC Driver API:** This supports the JDBC Manager-to-Driver Connection.
* The JDBC API uses a driver manager and database-specific drivers to provide connectivity to heterogeneous databases.
* The JDBC driver manager ensures that the correct driver is used to access each data source. The driver manager is capable of supporting multiple concurrent drivers connected to multiple heterogeneous databases.

Following is the architectural diagram, which shows the location of the driver manager with respect to the JDBC drivers and the Java application −



The **java.sql** package contains classes and interfaces for JDBC API. A list of popular *interfaces* of JDBC API are given below:

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

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

* DriverManager class
* Blob class
* Clob class
* Types class

## Common JDBC Components

The JDBC API provides the following interfaces and classes −

* **DriverManager:** This class manages a list of database drivers. Matches connection requests from the java application with the proper database driver using communication sub protocol. The first driver that recognizes a certain subprotocol under JDBC will be used to establish a database Connection.
* **Driver:** This interface handles the communications with the database server.
* **Connection:** This interface with all methods for contacting a database. The connection object represents communication context, i.e., all communication with database is through connection object only.
* **Statement:** You use objects created from this interface to submit the SQL statements to the database. Some derived interfaces accept parameters in addition to executing stored procedures.
* **ResultSet:** These objects hold data retrieved from a database after you execute an SQL query using Statement objects. It acts as an iterator to allow you to move through its data.
* **SQLException:** This class handles any errors that occur in a database application.

**5.3 Know about Connection interface and DriverManager Class**

**DriverManagerClass:**

The DriverManager class acts as an interface between user and drivers. It keeps track of the drivers that are available and handles establishing a connection between a database and the appropriate driver. The DriverManager class maintains a list of Driver classes that have registered themselves by calling the method DriverManager.registerDriver().

### Useful methods of DriverManager class

|  |  |
| --- | --- |
| **Method** | **Description** |
| 1) public static void registerDriver(Driver driver): | is used to register the given driver with DriverManager. |
| 2) public static void deregisterDriver(Driver driver): | is used to deregister the given driver (drop the driver from the list) with DriverManager. |
| 3) public static Connection getConnection(String url): | is used to establish the connection with the specified url. |
| 4) public static Connection getConnection(String url,String userName,String password): | is used to establish the connection with the specified url, username and password. |

**ConnectionInterface:**

A Connection is the session between java application and database. The Connection interface is a factory of Statement, PreparedStatement, and DatabaseMetaData i.e. object of Connection can be used to get the object of Statement and DatabaseMetaData. The Connection interface provide many methods for transaction management like commit(), rollback() etc.

**Methods of ConnectionInterface:**

**1) public Statement createStatement():** creates a statement object that can be used to execute SQL queries.

**2) public void commit():** saves the changes made since the previous commit/rollback permanent.

**3) public void rollback():** Drops all changes made since the previous commit/rollback.

**4) public void close():** closes the connection and Releases a JDBC resources immediately.

**5.4 List and know about JDBC drivers**

* What do you mean by driver?
  + Jdbc driver is a small piece of software that allows JDBC to connect to different databases.
* The driver is loaded with the help of a static method.



Every database has its own driver. The following table describes names for few databases:

|  |  |
| --- | --- |
| **DataBase name** | **Driver Name** |
| MS ACCESS | sun.jdbc.odbc.JdbcOdbcDriver |
| Oracle | Oracle.jdbc.driver.OdbcDriver |
| Microsoft SQL server 2000 | Com.Microsoft.sqlserver.jdbc.SqlServerDriver |
| MySQL | org.gjt.mm.mysql.Driver |

* All the JDBC drivers have been classified into 4-categories:

1. Type 1:JDBC ODBC bridge driver
2. Type 2: native-API/partly java driver
3. Type 3: Net-protocol driver.
4. Type 4: pure java driver.

**1: Type 1:JDBC ODBC bridge driver (or) Bridge Driver.**

* This driver provided by sun micro systems as the part of JDK.
* Internally, this driver will take support of ODBC(Open Databse Connectivity) driver to communicate with DataBase.
* **This driver converts all JDBC calls to ODBC and sends it to ODBC drivers. The ODBC driver converts ODBC calls into DataBase Specific calls.**

ODBC: Open Data Base Connectivity , it is a standard way for accessing databases & it is dependent of operating system.

|  |  |
| --- | --- |
| C:\Users\CME 1\Desktop\jdbc\jdbc-type1-driver.JPG | DSN: Data Source Name – keeps a record of which database needs to be accessed, the location of the database, & the driver needed to access the database.  There are 3-types of DSN:  a) user DSN-for a specific User.  b) System DSN- for all users on the machine.  c) File DSN- For all users who same drivers installed & the users can be on different machines. |

**Advantages of Type 1-driver:**

* It is very easy to use and maintain.
* Not required to install.
* Database Independent.

**.Disadvantages:**

* Performance is much low as JDBC calls ODBC & ODBC driver access database & then the result is retrieved.
* Platform dependent.
* Not suitable for web applications.
* Not support from JDK1.8 version onwards

**Type 2: native-API/partly java driver:**

|  |  |
| --- | --- |
| C:\Users\CME 1\Desktop\jdbc\jdbc-type2-driver.JPG | * This driver is exactly same as type-1 driver except that ODBC driver is replaced with database vendor specific native libraries(set of functions written in non-java). * **Type-2 driver converts JDBC calls into vendor specific native library calls which can be understandable directly database engine.**   **Advantages:**   * Offers better performance than type 1 driver.   **Dis advantages:**   * Libraries (API) needs to be installed on the client machine. * Not suitable for web applications. * Databse dependent. * Paltform dependent. |

**Type 3: Net-protocol driver.**

|  |  |
| --- | --- |
| C:\Users\CME 1\Desktop\jdbc\jdbc-type3-driver.JPG | * Type 3 drivers are written purely in java & used in a networked environment(3-tier architecture). * **The requests are routed to a middle tier which converts JDBC calls to database-specific calls.** * Middleware(IDS) can communicate with database using Type1 or Type2 or Type3 driver. * IDS(Internet Database Access Server) or middleware that allows you to create applications that can interact with back end DBMS.   **Advantages:**   * It can be used in web applications. * DataBase Independent. * Platform Independent.   **Disadvantages**:   * Maintenance of network protocol driver becomes costly because the database specific code has to be embedded into middle tier. * Network support is required on client machine. |

**Type 4: pure java driver (or) Thin driver (or) All java native protocol driver**

|  |  |
| --- | --- |
| C:\Users\CME 1\Desktop\jdbc\download.jpg | * This driver uses database specific native protocol to communicate with database. * This driver converts JDBC calls into DataBase specific calls directly.   **Advantages:**  1) it is platform independent .  2) its performance is very good.  **Disadvantages:**   * DataBase dependent. |

**5.5 Know about 3 kinds of Statements:**

**Statement , Prepared Statement and Callable Statement**

The connection (after being established) is used to send SQL statements to the database. There`re 3 types interfaces in java.sql package used for sending SQL statements to databases, namely **Statement** and its 2 sub interfaces **PreparedStatement** and **CallableStatement.** There methods of the connection objects are used to return objects of these statements.

* A **Statemen**t object is used to send a simple SQL statement to the database with no parameters. Its object returned by using the Connection object are used to return objects of these statements.

**Syntax:** **Statement stmt = con.createStatement();**

* A **Prepared** Statement object sends precompiled statements with or without IN parameters. Normally, we insert rows of data into the databases using the insert SQL statement. For example, we write an SQL statement which is sent to the database. If n rows need to be inserted, then the same statements get compiled n times. In Prepared Statement, only the values that have to be inserted are sent to the database again and again.

**Syntax:** **PreparedStatement ps= con.preparedStatement(String query);**

* A **CallableStatement** object is used to call stored procedures.

**Syntax:** **CallableSatement cs= com.prepareCall(String query);**

**5.6 UnderStand the steps in connecting to database using JDBC**

**Steps to connect to a database:**

1. Load the driver
2. Establish the connection.
3. Create the statements.
4. Execute query and obtain the results.
5. Iterate through the results.
6. Close the connection

**5.6.1 Know how to establish connection:**

* A connection to the database is established using the static method **getConnection(databaseURL)** of the **DriverManager** Class.
* The DriverManager class is a class for managing JDBC Drivers.
* The database URL takes the following form

**jdbc:subprotocol:subname**.

* If any problem occurs during accessing database, an SQLException is generated, else a connection reference is returned which refers to a connection to the database Connection is an interface in java.sql package.

**Syntax: Connection con= DriverManager.getconnection(databaseURL);**

* The following table shows the various database URLs for connecting to various databases.

|  |  |
| --- | --- |
| **DataBase** | **databaseURL** |
| MS ACCESS | jdbc:odbc:<DSN> |
| Oracle thin driver | jdbc:oracle:thin:@<HOST>:<PORT>:<SID> |
| MICROSOFT SQL SERVER | jdbc:Microsoft:sqlservver://<HOST>:<PORT>[;DATABASE NAME=<DB>] |

**A sample example program on establishing a connection:**

import java.sql.\*;

import java.io.\*;

public class example123

{

public static void main(String args[])throws SQLException

{

try

{

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver"); //loading of driver

Connection con=DriverManager.getConnection("jdbc:odbc:College"); //connection establishment

System.out.println("connection established");

}

catch(Exception e)

{

System.out.println(e);

}

}

}

**5.6.2 Create a statement :refer 5.5 tpoic**

Example Program:

import java.sql.\*;

import java.io.\*;

public class example123

{

public static void main(String args[])throws SQLException

{

try

{

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver"); //loading of driver

Connection con=DriverManager.getConnection("jdbc:odbc:College"); //connectionestablishment

System.out.println("connection established");

Statement st=con.createStatement();

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

while(rs.next()) //iterate through the result set

{

System.out.print(rs.getInt(1)+"\t");

System.out.print(rs.getString(2)+"\t");

System.out.print(rs.getInt(3)+"\t");

}

}

catch(Exception e)

{

System.out.println(e);

}

}

}

**5.6.3 Execute the Query:**

🡪The SQL statements are executed with the help of 3-methods.

* + ResultSet **executeQuery(**String sqlquery) throws SQLException.
  + int **executeUpdate**(String sqlquery) throws SQLException.
  + Boolean **execute**(String sqlquery) throws SQLException.

**1. executeQuery():**

* It is used for executing SQL statements that returns a single ResultSet, e.g: select statement.
* The rows fetched from database are returned as a single ResultSet object.

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

**2. executeUpdate():**

* It is used for DDL, DML SQL statements like insert, update, delete & create.
* This method returns an integer value for DML to indicate the number of rows affected/inserted and 0-for DDL statements which don’t return anything.

**PreparedStatement ps=con.prepareStatement("insert into student values (?,?,?,?,?)”);**

* The values of IN(?) is sent using:

ps.setInt(1,1000);

ps.setString(2,”abc”);

ps.executeUpdate();

**3 Execute() method:**

It is used for callable statement when the statement may return more than one ResultSet or update counts or combination of both. This happens when stored procedures are executed.

Example program:

import java.sql.\*;

import java.io.\*;

public class example963

{

public static void main(String args[])throws SQLException

{

try

{

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver"); //loading of driver

Connection con=DriverManager.getConnection("jdbc:odbc:College"); //connection establishment

System.out.println("connection established");

Statement st=con.createStatement(); //statement creation

ResultSet rs=st.executeQuery("select \* from emp"); //executing simple query

ResultSetMetaData rsmd=rs.getMetaData(); //getting metadata of emp table

int c=rsmd.getColumnCount();

System.out.println("no.of column in result set:"+c);

while(rs.next()) //iterate through the result set

{

System.out.print(rs.getInt(1)+"\t");

System.out.print(rs.getString(2)+"\t");

System.out.print(rs.getInt(3)+"\t");

System.out.println();

}

con.commit();

PreparedStatement ps=con.prepareStatement("insert into emp values(4,'lakshman',35250)"); //prepareStatement

System.out.println(ps.executeUpdate());

con.close();

}

catch(Exception e)

{

System.out.println(e);

}

}

**}**

**5.6.4 Process the ResultObject**

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

Resultset rs =st.executeQuery(sql query);

Ex: Resultset rs =st.executeQuery(select \*from student);

**Iterate through the results.**

* The resultset is iterated with the help of method ‘next’ which returns a Boolean value to indicate that the ResultSet has more rows to be iterated.
* The next method moves the cursor to the next row.
* The individual column data is obtained by using accessor i.e., getX-method

while(rs.next())

{

System.out.print(rs.getInt(1)+"\t");

System.out.print(rs.getString(2)+"\t");

}

**Types of Resultsets**

**ResultSet can be divided into 3 categories**

* Based on operations
* ReadOnly ResultSet
* Updatable ResultSet
* Based on cursor movement
* Non-Scrollable ResultSet in JDBC
* Scrollable ResultSet

By default a ResultSet Interface is Non-Scrollable, In non-scrollable ResultSet we can move only in forward direction (that means from first record to last record), but not in Backward Direction, If you want to move in backward direction use **Scrollable Interface**.

**Difference between Scrollable ResultSet and Non-Scrollable ResultSet**

|  |  |  |
| --- | --- | --- |
|  | **Non-Scrollable ResultSet** | **Scrollable ResultSet** |
| 1 | Cursor move only in forward direction | Cursor can move both forward and backward direction |
| 2 | Slow performance, If we want to move nth record then we need to n+1 iteration | Fast performance, directly move on any record. |
| 3 | Non-Scrollable ResultSet cursor can not move randomly | Scrollable ResultSet cursor can move randomly |

**Create Scrollable ResultSet:** To create a Scrollable ResultSet, create Statement object with two parameters.

**Syntax:** Statement stmt=con.CreateStatement(param1, param2); // parm1 type and param2 mode

These type and mode are predefined in ResultSet Interface of Jdbc like below which is static final.

**Type:**

* + public static final int TYPE\_FORWARD\_ONLY
  + public static final int TYPE\_SCROLL\_INSENSITIVE
  + public static final int TYPE\_SCROLL\_SENSITIVE

**Mode:**

* + public static final int CONCUR\_READ\_ONLY
  + public static final int CONCUR\_UPDATABLE

**Methods:** It has the following methods.

|  |  |
| --- | --- |
| **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 boolean first():** | is used to move the cursor to the first row in result set object. |
| **4) public boolean last():** | is used to move the cursor to the last row in result set object. |
| **5) public boolean absolute(int row):** | is used to move the cursor to the specified row number in the ResultSet object. |
| **6) public boolean relative(int row):** | is used to move the cursor to the relative row number in the ResultSet object, it may be positive or negative. |
| **7) public int getInt(int columnIndex):** | is used to return the data of specified column index of the current row as int. |
| **8) public int getInt(String columnName):** | is used to return the data of specified column name of the current row as int. |
| **9) public String getString(int columnIndex):** | is used to return the data of specified column index of the current row as String. |
| **10) public String getString(String columnName):** | is used to return the data of specified column name of the current row as String. |

**Sample program on Scrollable ResultSet:**

import java.sql.\*;

public class Rs

{

public static void main(String[] args)

{

try

{

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

Connection conn = DriverManager.getConnection("jdbc:odbc:STUDENT");

Statement stmt=conn.createStatement(ResultSet.TYPE\_SCROLL\_INSENSITIVE,ResultSet.CONCUR\_UPDATABLE);

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

while (rs.next())

{

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

}

System.out.println(" the first record is");

System.out.println(rs.first());

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

System.out.println(" the last record is");

System.out.println(rs.last());

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

System.out.println(" the previous record is");

System.out.println(rs.previous());

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

System.out.println(" the next record is");

System.out.println(rs.next());

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

System.out.println(" the obsolute position is");

rs.absolute(4);

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

rs.previous();

rs.relative(-1);

System.out.println("second RECORD...");

System.out.println(rs.getInt("SID")+" "+rs.getString("SNAME")+" "+rs.getString("AGE"));

}

catch(Exception e)

{

System.out.println(e);

}

}

}

**ResultSetMetaData object**:

* This object is used to obtain metadata about the ResultSet that include number of columns, types of columns……..
* The method **getColumnCount()** –method returns number of columns in the ResultSet.
* The method **getColumnTypeName(int index)-**returns type of the data the column holds

**5.6.5 close the connection:**

**public void close():** closes the connection and Releases a JDBC resources immediately.

Ex: con.close();

**5.7. Refer 5.6.3 (3) program**