

UNIT 5

Introduction to Database Management Systems and JDBC:

when we develop a program, we need to concentrate on 3 things.

1. user interface, 2. data storage and 3. process.

In traditional programming, we manage all three things with same language. but if the data storage is managed with language in the form of files, it leads to many problems like no proper data sharing over network, no data consistency, no data integrity, no data security etc. to overcome these problems, DBMS products are designed and being used.

When the data is managed with DBMS product, the data is stored and managed in tabular format. we use the language SQL to manage data. the basic operations a programmer is to identify are called CRUD operations. Create data, Read data, Update data and Delete data. we use insert, select, update and delete queries for CRUD operations.

Once data is managed independent of UI and process, data gets more security. Once data is managed in DBMS, we need to connect from our program to the database to get and manipulate data in program. Once an applications UI and process is managed with a language and data is managed with a DBMS, then as there are 2 LAYERS (tiers) of software is used for application management, it is called a 2-tier architecture. As the DBMS is at rear part of the app and the UI is at front part that is visible to user, the program is called Front End and the DBMS is called Back End of the application.

The FE can be designed with any RAD tool (Rapid Application Development) like VB, VB.Net, C#, Java, VC++ etc. the back end can be any DBMS product like oracle, ms-sql server, mysql, sybase, ingrece etc.

If the app is designed as web-app, then the UI is managed with HTML, DHTML, java script etc , process is managed with PHP, python, java, c# etc and data with a DBMS product, then it is called a 3-T architecture.

The process of connecting java with a DBMS is called JDBC-ODBC architecture.

Q. Explain JDBC-ODBC architecture

Types of Drivers:

For establishing connection between a program and DBMS, we take help of appropriate drivers. The DB drivers are different types :

1. JDBC ODBC bridge : it was provided free with java software. but it is not used now a days (from jdk9 version onwards, even windows 7 64 bit OS onwards the OS also doesn't support that.)
2. Pure Java driver : these are from SUN micro systems, but need to be purchased.
3. Net Oriented drivers: these are 3rd party drivers, need to be purchased.
4. Thin drivers / type 4 drivers: these are provided free of cost by the DB product companies like... oracle has oracle thin driver, mysql has my sql j-connector etc.

oracle thin driver is a part of oracle software, but mysql jconnector need to be downloaded separately.

classpath : when using DB drivers, the driver files are generally given with extension .jar.

ex: ojdbc14.jar

Once these files are installed / placed / copied into a folder in system, as, when running java code, the JRE doesn't know the location of classes of these jar files. so we must set something called "classpath" with OS, so that the JRE can pick up required classes from this location.

for setting classpath...

right click on "this pc" -> properties -> advanced system setting -> system properties dialog raises, in which select advanced tab -> click on "environment variables", in user variables (top portion) check for classpath, if not existing, then click on "new" button, a dialog raises, in which give variables name "classpath" and mention the jar file entire path in value box (place a . ; and paste the path of jar file). if classpath already exists, then select that, click on edit button, click on "new" value button, place the classpath of jar file.

ex:

generally for mysql (if the driver is in d:\):

```
.;D:\mysql-connector-j-9.7.0_1\mysql-connector-j-9.7.0\mysql-connector-j-9.7.0.jar
```

for oracle :

```
.; C:\oracle\app\oracle\product\10.2.0\server\jdbc\lib\ojdbc14.jar
```

in above all the class path includes a DOT (.) at beginning, that refers current folder.

Q. Explain different types of JDBC drivers.

Establishing Connectivity with Databases –

Loading Drivers, Creating Connections:

For connecting java with any DB, we need to follow following steps.

1. Load drivers into memory :
2. Establish connection
3. Create statements
4. Execute queries

1. Load Drivers into memory: The class "java.lang.Class" has a static method with name `forName()`, that holds the drivers / loads drivers into memory.

The classes, are jar files that are to be loaded are argumented to `forName()` method, and the code would be :

```
oracle :      Class.forName("oracle.jdbc.OracleDriver");
mysql:       Class.forName("com.mysql.cj.jdbc.Driver");
H2 database : Class.forName("org.h2.Driver");
and so on.
```

** for mysql, this loading drivers is optional in latest versions. (it is automatically loaded).

2. Establishing connection from java to DB

The class "java.sql.DriverManager" has a method getConnection(), that establishes connection to the DB specified and returns the connection as an instance of java.sql.Connection interface. so the code would be :

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

The getConnection() takes 3 arguments, the DB URL, User ID and Password.

The Oracle's DB URL :- "jdbc:oracle:thin:@localhost:1521:xe"

The xe at end is for oracle 10g express edition, for some versions / editions it could also be orcl or any customized DB name.

Ex:

```
Connection con=
```

```
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe",  
"user name","password");
```

generally user name is system / scott.

The MySQL DB URL :- "jdbc:mysql://localhost:3306/DB_Name"

Ex:

```
Connection con=
```

```
DriverManager.getConnection("jdbc:mysql://localhost:3306/DB_Name",  
"username","password");
```

generally user name is root

The H2 DB (personal DB), the DB URL : "jdbc:h2:~/test"

Ex:

```
Connection con=DriverManager.getConnection(  
"jdbc:h2:~/test","user name","password");
```

generally user name is "sa" and password is empty ""

once connection is established, we create java statements and execute queries.

Q. Explain loading drivers and establishing JDBC-ODBC connections

JDBC Classes and Interfaces

Class : java.lang package has a class with name “Class”, that has a static method `forName()`, that is used to load drivers into memory, that can be further used to establish connection.

Ex:

```
Class.forName(“Driver Class Name”);
```

DriverManager : This class is defined in java.sql package and is used to establish connection from java program to the DB whose URL is specified, using above loaded drivers. The static method `getConnection()` is used to establish connection, and returns the connection as an instance of `Connection` interface.

Code :

```
Connection con = DriverManager.getConnection(“URL”, “UN”, “PWD”);
```

Connection : This interface is defined in java.sql package and used to hold the connection established and used to create either `Statement` or `PreparedStatement` interface instances. It has methods `createStatement()` that creates instance of `Statement` interface and `prepareStatement()` that creates instance of `PreparedStatement`.

Ex:

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

Or

```
PreparedStatement pst=con.prepareStatement(“string query”);
```

Statement / PreparedStatement : these are defined in java.sql package, and are used to compile string query into SQL statement and execute on DB.

Methods of these two objects are :

`executeQuery()` : used to execute a select query and this method returns no. of rows of data fetched from DB, as instance of `ResultSet`.

`executeUpdate()` : Mused to execute DML queries like create , update and delete. This method returns an integer value, specifiying no. of rows affected with the query.

The difference between the statement and prepared statement is that ,the `Statement` object is first created as empty object and the queries are passed while executing queries, whereas the `PreparedStatement` is created along with query and the methods are simply called without arguments.

So, when using statement object, the string query is compiled into SQL statement every time it is executed, whereas, in PreparedStatement, the string query is compiled as SQL statement when the instance is created, and NOT compiled everytime when it is executed.

Ex:

```
Statement st=con.createStatement();  
ResultSet rs=st.executeQuery("select * from emp");
```

OR

```
PreparedStatement pst=con.prepareStatement("select * from emp");  
ResultSet rs=st.executeQuery();
```

The PreparedStatement provides a facility of mentioning PLACE HOLDERS with a question mark symbol, that can be replaced with values later with the methods setInt(), setString() etc.

ResultSet: This interface is instantiated with the executeQuery() method of Statement or PreparedStatement. This works like a pointer pointing to each row of the fetched data with select query.

It has methods first(), next(), previous() and last() to move across each row of data. These methods return boolean based on existence of data. And the methods getInt(), getString() etc are used to fetch data of each column / field of current row.

```
While( rs.next())  
    System.out.println( rs.getInt(1)+" "+ rs.getString(2) );
```

Executing CRUD Operations using SQL.

Q. Explain the classes and interfaces related with DB connectivity.

These two are Practicals. So no theory is required.

Handling Database Data in GUI Applications.

Developing Simple Database Applications using Java and JDBC.