Advanced Java Programming

Unit-2

Database Connectivity

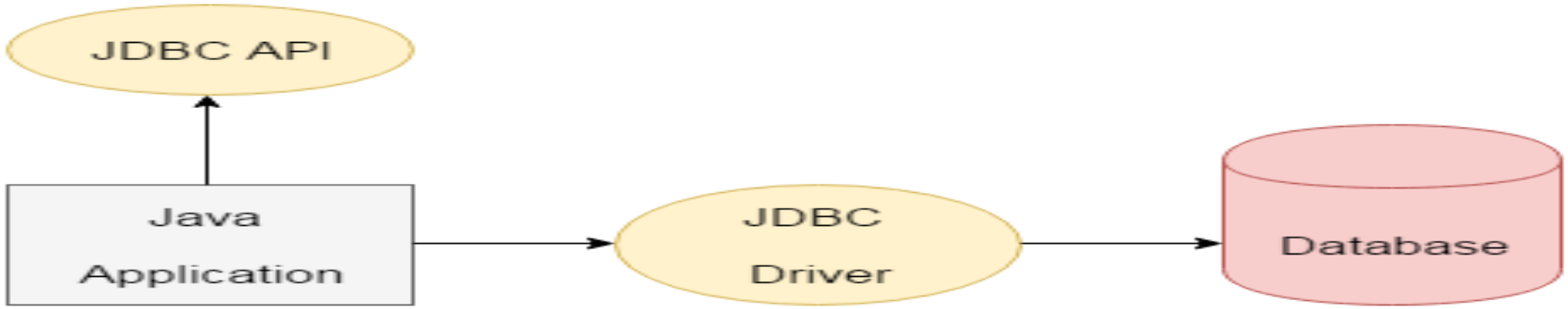
What is JDBC?

- JDBC is a Java standard that provides the interface for connecting from Java to relational databases.
- The JDBC standard is defined by Sun Microsystems and implemented through the standard JDBC API.
- JDBC stands for Java Database Connectivity, which is a standard Java API for database independent connectivity between the Java programming language and a wide range of databases.
- We can use JDBC API to handle database using Java program and can perform the following activities

  - Connect to the database

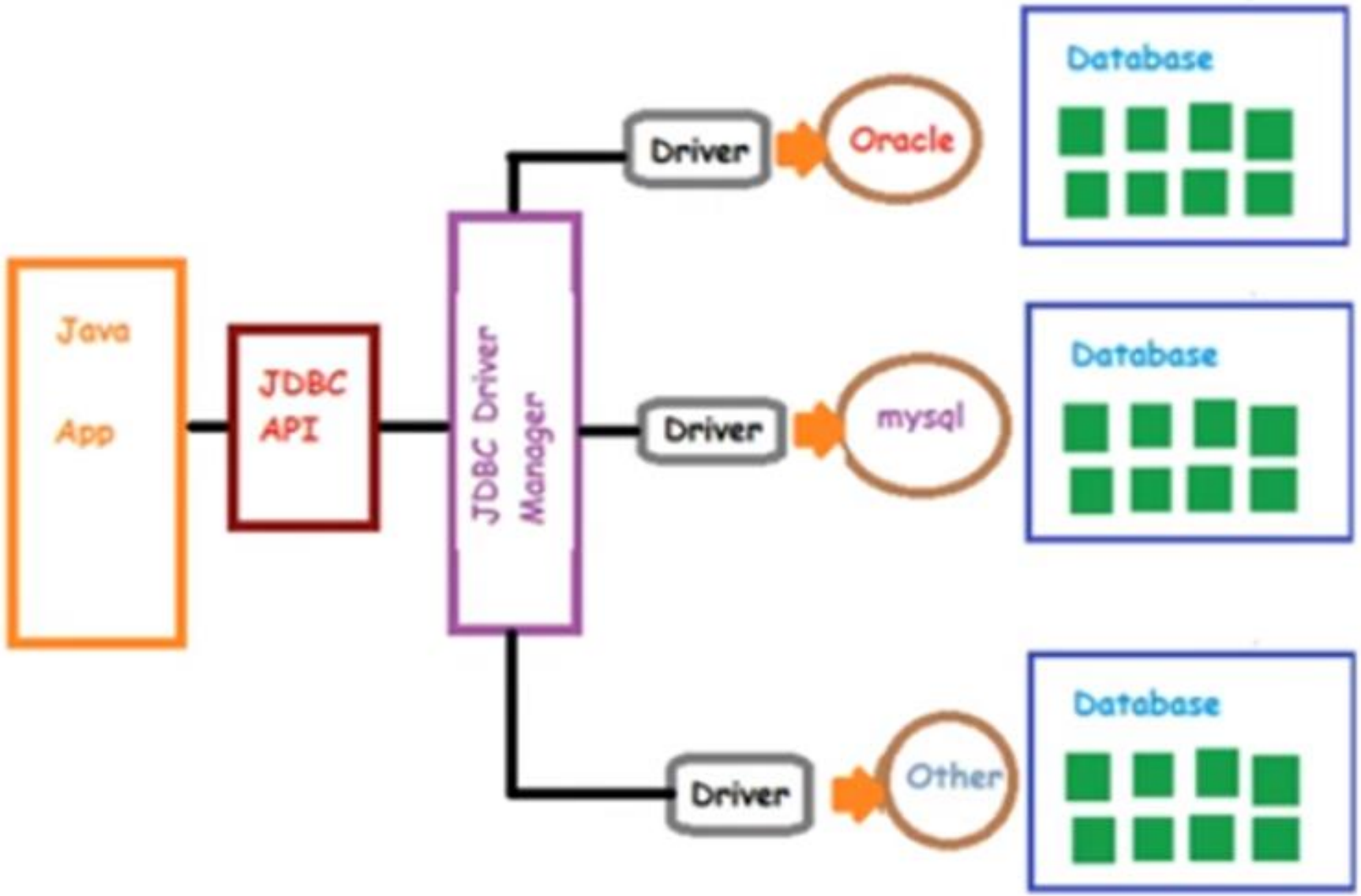
  - Execute queries and update statements to the database

  - Retrieve the result received from the database.



JDBC architecture.

- The JDBC API uses a Driver Manager and database precise drivers to provide clear connectivity to heterogeneous databases
- The Driver Manager is capable of supporting multiple concurrent drivers connected to multiple heterogeneous databases.
- The **JDBC API** is the top layer and is the programming interface in Java to structured query language (SQL) which is the standard for accessing relational databases.
- The JDBC API communicates with the **JDBC Driver Manager API**, sending it various SQL statements.
- The manager communicates with the various third party drivers that actually connect to the database and return the information from the query.


Java
App
JDBC
API
JDBC Driver
Manager
Driver
Oracle
Database
Driver
mysql
Database
Driver
Other
Database

Components of JDBC.

- Following is list of common JDBC Components:

DriverManager :

- This class manages a list of database drivers. It matches connection requests from the java application with the proper database driver using communication sub protocol.

Driver:

- This interface handles the communications with the database server. We will interact directly with Driver objects very rarely. Instead, we use DriverManager objects, which manage objects of this type.

Connection:

- This interface with all methods for contacting a database. The connection object represents communication context

Statement:

- We use objects created from this interface to submit the SQL statements to the database.

ResultSet:

- These objects hold data retrieved from a database after we execute an SQL query using Statement objects

SQLException:

- This class handles any errors that occur in a database application.

JDBC Drivers.

- JDBC Driver is a software component that enables java application to interact with the database.
- There are four types of JDBC drivers
  - **Type 1- JDBC-ODBC Bridge Driver:**
  - **Type 2 - Native-API/ Partly Java Driver:**
  - **Type 3- Network Protocol driver**
  - **Type 4 – Thin driver**

Steps to connect to the database in Java

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.
- i.e Class.forName(**string**);

Example

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

2. **Creating connection**

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

i.e, **Connection con = DriverManager.getConnection(db_url,user_name,password)**

Example

DriverManager.getConnection("jdbc:mysql://localhost/dbname","root",“ ");

3. Creating statement

- The **createStatement()** method of Connection interface is used to create statement.

Example

Statement stmt = con.createStatement();

4. Executing queries

- Execute SQL statement with one of its three execute methods.
- **execute():** can run both select and insert/update/delete statements.
- **executeQuery():** can executes statements that returns result set. i.e., select statements.
- **executeUpdate()**: method execute sql statement that insert/update/delete data at the database

5. Getting Result

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

6. Closing connection

```
Con.close();
```

```
import java.sql.*;
public class DBconnectionEx {
    public static void main(String [] arg)
    {
        try{
            //step 1
            Class.forName("com.mysql.jdbc.Driver");
            //step  2
            Connection con=DriverManager.getConnection("jdbc:mysql://localhost/Company","root","");
            //step 3
            Statement stmt=con.createStatement();
            //step 4
            ResultSet rs=stmt.executeQuery("select * from Employee");
```

```
    System.out.println("Eno"+"   "+"Ename"+"     "+"Eno");
     //step 5
          while(rs.next())
          {
              System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"   "+rs.getString(3));
          }
          //step 6
          con.close();
         }catch(Exception e)
        {
          e.printStackTrace();
        }
    }
}
```

Example for Insertion:

```
import java.sql.*;
public class DBconnectionEx {
    public static void main(String [] arg)
    {
        try{
          Class.forName("com.mysql.jdbc.Driver");
           Connection con=DriverManager.getConnection("jdbc:mysql://localhost/Company","root","");
           Statement stmt=con.createStatement();
           int i=stmt.executeUpdate("insert into Employee values(108,'Samar','kathmandu')");
           System.out.print(i+" row is insert successfully");

           con.close();
         }catch(Exception e){e.printStackTrace();}
}
}
```

Statement in JDBC

- A statement in JDBC is used to execute SQL queries after connecting to the database.
- Three ways of executing a query:
  - Standard Statement
  - Prepared Statement
  - Callable Statement

Standard Statement

- Statement objects are never instantiated directly.
- To do so, the **createStatement ()** of Connection is used.

  Statement stmt=dbcon.createStatement ()

  ResultSet rs=stmt.executeQuery (“Select * from student”);
- A query that returns data can be executed using executeQuery () of Statement.
- This method executes the statement and returns ResultSet object that contains the resulting data
- For inserts, updates, deletes use executeUpdate ().

Prepared Statement:

- The PreparedStatement interface is a sub-interface of Statement. It is used to execute parameterized query.
- PreparedStatement object is an SQL statement that is pre-compiled and stored.
- This object can then be executed multiple times much more efficiently than preparing and issuing the same statement each time it is needed.
- The **prepareStatement ()** method of Connection interface is used to return the object of PreparedStatement
- The question marks represent dynamic query parameters. These parameters can be changed each time the prepared statement is called.
- Before a PreparedStatement object is executed, the value of each**?** Parameter must be set.
- This is done by calling a **setType()** method, where Type is the appropriate type for the parameter
- The first argument to the **setType()** methods is the ordinal position of the parameter to be set, with numbering starting at 1. The second argument is the value to which the parameter is to be set.

Example of PreparedStatement interface that inserts the record:

```
import java.sql.*;
public class DBconnectionEx {
    public static void main(String [] arg)
    {
        try{
            Class.forName("com.mysql.jdbc.Driver");
           Connection con=DriverManager.getConnection("jdbc:mysql://localhost/Company","root","");
           PreparedStatement stmt=con.prepareStatement("insert into Employee values(?,?,?)");
            stmt.setInt(1,150);//1 specifies the first parameter in the query
            stmt.setString(2,"Ratan");
            stmt.setString(3, "pokhara");
            int i=stmt.executeUpdate();
            System.out.println(i+" records inserted");
           con.close();
         }catch(Exception e){ e.printStackTrace(); }
    }
}
```

ResultSetMetaData.

- JDBC Meta Data is the collective information about the data structure and property of a column available in table.
- The Metadata of any table tells us the name of the columns, datatype used in column and constraint used to enter the value of data into column of the table.
- It provides following methods.

executeQuery ( ): The method retrieves a record set from a table in database. The retrieve record set is assigned to a result set.

getMetaData ( ): The Result Set call gets Metadata ( ), which return us the property of the retrieve record set (length, field, and column).Meta Data account for data element and its attribute.

getcolumncount ( ): The method returns us an integer data type and provides you the number of column in the Result set object.

Example:

```
import java.sql.*;
public class DBconnectionEx {
    public static void main(String [] arg)
    {
        try{
          Class.forName("com.mysql.jdbc.Driver");
           Connection   con=DriverManager.getConnection("jdbc:mysql://localhost/Company","root","");
           Statement stmt=con.createStatement();
           ResultSet rs=stmt.executeQuery("select * from Employee");
           ResultSetMetaData md=rs.getMetaData();
           int col=md.getColumnCount();
           System.out.println("Table Name:"+md.getTableName(1));
           for(int i=1;i<=col;i++)
               System.out.printf("%-8s\t",md.getColumnName(i));
               System.out.println();
```

```
while(rs.next())
        {
          for(int i=1;i<=col;i++)
              System.out.printf("%-8s\t",rs.getObject(i));
              System.out.println();
        }
        con.close();
      }catch(Exception e)
      {
        e.printStackTrace();
      }
   }

}
```

Table Name:employee

Eno	Ename	Eaddress
101	Ramesh	Kathmandu
102	Harish	Bhaktapur
103	Taveeta	Lalitpur
201	Sapana	Bharatpur
108	Samar	kathmandu
301	Kamal	mahendranagar
150	Ratan	pokhara

Scrollable and Updatable Result Sets

- In a *scrollable* result, you can move forward and backward through a result set and even jump to any position.
- In an *updatable* result set, you can programmatically update entries so that the database is automatically updated.

Scrollable Result Sets:

- By default, result sets are not scrollable or updatable
- To obtain scrollable result sets, obtain a different **Statement** object with the method

  Statement stat = conn.createStatement(type, concurrency);

- For a prepared statement, use the call

  PreparedStatement stat = conn.prepareStatement(sql, type,concurrency);

ResultSet type:

TYPE_FORWARD_ONLY: The result set in not scrollable(default).

TYPE_SCROLL_INSENSITIVE:: The result set is scrollable but not sensitive to database changes.

TYPE_SCROLL_SENSITIVE: The result set is scrollable and sensitive to database changes

ResultSet concurrency:

CONCUR_READ_ONLY: The result set cannot be used to update the database.

CONCUR_UPDATABLE: The result set can be used to update the database.

Example:

```
import java.sql.*;
class FetchRecord{
public static void main(String args[])throws Exception{

Class.forName("com.mysql.jdbc.Driver");
Connection con=DriverManager.getConnection("jdbc:mysql://localhost/College","root","");

Statement stmt=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);
ResultSet rs=stmt.executeQuery("select * from student");

//getting the record of 3rd row
rs.absolute(3);
System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));

con.close();
}}
```

Row Sets

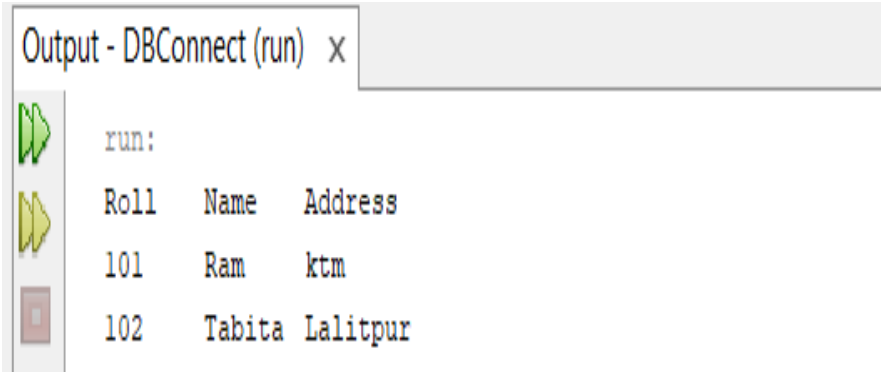
- RowSet is an interface available under javax.sql.RowSet.
- It is sub-type of ResultSet.
- RowSet functionality is similar to ResultSet.

ResultSet	RowSet
ResultSet store the result return by select query.	RowSet store the result returned by the select query.
By-default ResultSet is Forward-only and Read-only.	By-Default RowSet is Scrollable and Updatable.
ResultSet is connection oriented i.e you can access the ResultSet as long as connection is available.	RowSet is connection or connection less, we can access RowSet without Connection.
ResultSet Object is not serialization	RowSet object is serialization

- The implementation classes of the RowSet interface are as follows:
  - JdbcRowSet
  - CachedRowSet
  - WebRowSet
  - JoinRowSet
  - FilteredRowSet
- The **advantages** of using RowSet are given below:
  - It is easy and flexible to use.
  - It is Scrollable and Updatable by default.

Example: RowSet

```
import javax.sql.*;
import javax.sql.rowset.*;
public class RowsetEx {
    public static void main(String[] args) throws Exception {
        Class.forName("com.mysql.jdbc.Driver");
        //Creating and Executing RowSet
        JdbcRowSet rowSet = RowSetProvider.newFactory().createJdbcRowSet();
        rowSet.setUrl("jdbc:mysql://localhost/College");
        rowSet.setUsername("root");
        rowSet.setPassword("");
        rowSet.setCommand("select * from Student");
        rowSet.execute();
        System.out.println("Roll\tName\tAddress");
        while (rowSet.next()) {
            System.out.println(rowSet.getInt(1)+"\t"+rowSet.getString(2)+"\t"+rowSet.getString(3));
        }
}}
```



CachedRowSet:

- It is the child interface of RowSet.
- It is by default scrollable and updatable.
- It is disconnected RowSet. i.e, we can use RowSet without having database connection
- It is serializable
- The main advantage of CachedRowSet is we can send this RowSet object for multiple people across the network and all those people can access RowSet data without having DB connection.
- If we perform any update operation (like insert, delete, update) to the CachedRowSet, to reflect those change Compulsory connection should be established
- Once Connection established then only those changes will be reflected in Database.
- Example below demonstrate how CahedRowSet work?

```
import java.sql.*;
import javax.sql.rowset.*;
public class CachedRowEx {
    public static void main(String [] args)throws Exception
    {
        Class.forName("com.mysql.jdbc.Driver");
        Connection con= DriverManager.getConnection("jdbc:mysql://localhost/HCOE","root","");
        String sql="select * from student";
        Statement st=con.createStatement();
        ResultSet rs=st.executeQuery(sql);
        RowSetFactory rsf= RowSetProvider.newFactory();
        CachedRowSet crs=rsf.createCachedRowSet();
        crs.populate(rs);
        con.close();
        //read data from CachedRowSet
        System.out.println("Roll\tName\tAddress");
        while(crs.next())
            System.out.println(crs.getInt(1)+"\t"+crs.getString(2)+"\t"+crs.getString(3));
    }}
```