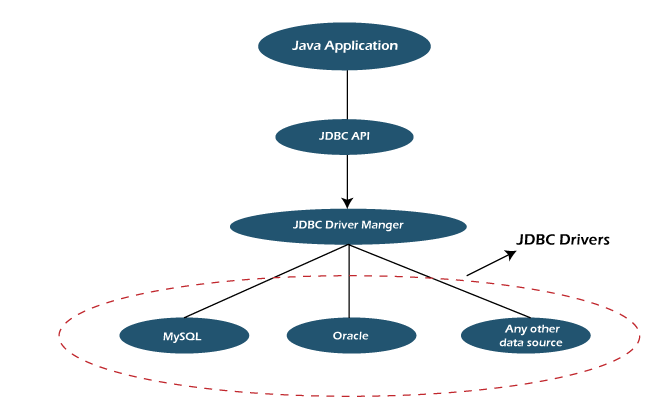
UNIT-III: Working with Database

Java Database Connectivity (JDBC) is an Application Programming Interface (API), from Sun microsystem that is used by the Java application to communicate with the relational databases from different vendors. JDBC and database drivers work in tandem to access spreadsheets and databases. Design of JDBC defines the components of JDBC, which is used for connecting to the database.

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.

javax.sql.\*;

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.

**Architecture of JDBC**

1) Application: It is the Java servlet or an applet that communicates with the data source.

2) The JDBC API: It allows the Java programs to perform the execution of the SQL statements and then get the results.

A few of the crucial interfaces and classes defined in the JDBC API are the following:

* Drivers
* DriverManager
* Statement
* Connection
* PreparedStatement
* ResultSet

3) DriverManager: DriverManager plays a crucial role in the architecture of JDBC.It uses database-specific drivers to connect the enterprise applications to various databases.

4) JDBC drivers: To interact with a data source with the help of the JDBC, one needs a JDBC driver which conveniently interacts with the respective data source.

**Different Types of Architecture of JDBC**

The architecture of the JDBC consists of two and three tiers model in order to access the given database.

**Two-tier model:** In this model, the application interacts directly with the source of data. The JDBC driver establishes the interaction between the data source and the application. When a query is sent by the user to the data source, the reply of those sent queries is sent directly to the user.

The source of data can be located on a different machine, and that machine is connected to the user machine following a client-server paradigm, where the machine which is sending the query is the client machine, and the machine that is sending the result of those queries is acting as the server.

**Three-tier model:** In this model, the queries of the user are being sent to the middle-tier services, from where the commands are sent again to the source of data. The answers to those queries are reverted to the middle tier, and from there, it is again sent to the user.

**Working with JDBC APIs:**

**Connection interface:** A *Connection* is a session between a Java application and a specific database. SQL statements are executed and results are returned within the context of a connection.

A *Connection* object's database is able to provide information describing its tables, its supported SQL grammar, its stored procedures, the capabilities of this connection, and so on. This information is obtained with the *getMetaData* method.

**Connection Interface Commonly used Methods**

* *public Statement createStatement() -*creates a *statement* object that can be used to execute SQL queries.
* *public Statement createStatement(int resultSetType,int resultSetConcurrency)* - Creates a Statement object that will generate ResultSet objects with the given type and concurrency.
* *public void setAutoCommit(boolean status)* - is used to set the commit s status. By default it is true.
* *public void commit() -*saves the changes made since the previous commit/rollback permanent.
* *public void rollback()  -* Drops all changes made since the previous commit/rollback.
* *public void close() -*closes the connection and releases JDBC resources immediately.

**Statement interface Overview**

* Statement interface object used for executing a static SQL statement and returning the results it produces.Statement interface extends Wrapper, AutoCloseable interfaces.
* By default, only one ResultSet object per Statement object can be open at the same time. Therefore, if the reading of one ResultSet object is interleaved with the reading of another, each must have been generated by different Statement objects.
* All execution methods in the Statement interface implicitly close a current ResultSet object of the statement if an open one exists.

**Statement interface Commonly used methods**

The important methods of Statement interface are as follows:

* *boolean execute(String sql)* - Executes the given SQL statement, which may return multiple results.
* *int[] executeBatch()* - Submits a batch of commands to the database for execution and if all commands execute successfully, returns an array of update counts.
* *ResultSet executeQuery(String sql)* - Executes the given SQL statement, which returns a single ResultSet object.
* *int executeUpdate(String sql)* - Executes the given SQL statement, which may be an INSERT, UPDATE, or DELETE statement or an SQL statement that returns nothing, such as an SQL DDL statement.

**PreparedStatements**

Sometimes it is more convenient to use a *PreparedStatement* object for sending SQL statements to the database. This special type of statement is derived from the more general class, Statement.

The PreparedStatement's primary features are:

* Easy to insert parameters into the SQL statement.
* Easy to reuse the PreparedStatement with new parameters.
* May increase performance of executed statements.
* Enables easier batch updates.

**PreparedStatement interface**

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

*PreparedStatement* interface takes parameters and the advantage of using SQL statements that take parameters is that you can use the same statement and supply it with different values each time you execute it.

**PreparedStatement interface Commonly used methods**

The important methods of PreparedStatement interface are given below:

* *public void setInt(int paramIndex, int value*) - sets the integer value to the given parameter index.
* *public void setString(int paramIndex, String value)* - sets the String value to the given parameter index.
* *public void setFloat(int paramIndex, float value)* - sets the float value to the given parameter index.
* *public void setDouble(int paramIndex, double value)* - sets the double value to the given parameter index.
* *public int executeUpdate()* - executes the query. It is used to create, drop, insert, update, delete etc.
* *public ResultSet executeQuery()* - executes the select query. It returns an instance of ResultSet.

**ResultSet Interface**

The ResultSet interface provides methods for retrieving and manipulating the results of executed queries, and ResultSet objects can have different functionality and characteristics.

**Commonly used methods of ResultSet interface**

* public boolean next(): is used to move the cursor to the one row next from the current position.
* public boolean previous(): is used to move the cursor to the one row previous from the current position.
* public boolean first(): is used to move the cursor to the first row in result set object.
* public boolean last(): is used to move the cursor to the last row in result set object.
* public int getInt(int columnIndex): is used to return the data of specified column index of the current row as int.
* public int getInt(String columnName): is used to return the data of specified column name of the current row as int.
* public String getString(int columnIndex): is used to return the data of specified column index of the current row as String.
* public String getString(String columnName): is used to return the data of specified column name of the current row as String.
* public boolean absolute(int row): is used to move the cursor to the specified row number in the ResultSet object.

**Java Database Connectivity with MySQL**

To connect Java application with the MySQL database, we need to follow 5 following steps.

In this example we are using MySql as the database. So we need to know following informations for the mysql database:

1. **Driver class:**The driver class for the mysql database is **com.mysql.jdbc.Driver**.
2. **Connection URL:**The connection URL for the mysql database is **jdbc:mysql://localhost:3306/cai** where jdbc is the API, mysql is the database, localhost is the server name on which mysql is running, we may also use IP address, 3306 is the port number and cai is the database name. We may use any database, in such case, we need to replace the cai with our database name.
3. **Username:**The default username for the mysql database is **root**.
4. **Password:**It is the password given by the user at the time of installing the mysql database. In this example, we are going to use root as the password.

**#1.Establishing a connection eith MySQL Server**

import java.sql.\*;

class MysqlConnection

{

public static void main(String args[])

{

try

{

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

Connection con=

DriverManager.getConnection("jdbc:mysql://localhost:3306","root","root");

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

con.close();

}

catch(Exception e){ System.out.println("Unable to establish Connection ");}

}

}

**#2.Creating a database in MySQL Server**

import java.sql.\*;

class MysqlConnection

{

public static void main(String args[])

{

try

{

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

Connection con=

DriverManager.getConnection("jdbc:mysql://localhost:3306","root","root");

Statement stmt=con.createStatement();

stmt.executeUpdate("create database vsem");

System.out.println("database vsem created");

con.close();

}

catch(Exception e){ System.out.println("unable to create database");}

}

}

**#3.Creating a table in a database**

import java.sql.\*;

class MysqlConnection

{

public static void main(String args[])

{

try

{

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

Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/vsem","root","root");

Statement stmt=con.createStatement();

stmt.executeUpdate("create table std (rno varchar(10),name varchar(20))");

System.out.println("Table created");

con.close();

}

catch(Exception e){ System.out.println("unable to create table");}

}

}

**#4.Inserting data into the table by accepting input from the user**

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.Statement;

import java.util.\*;

class MysqlConnection

{

public static void main(String args[])

{

Scanner sc = new Scanner(System.in);

try

{

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

Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/vsem","root","root");

System.out.println("Enter Roll No:");

String a=sc.next();

System.out.println("Enter Name:");

String b = sc.next();

Statement stmt=con.createStatement();

String sql="insert into std values('"+a+"','"+b+"')";

int i = stmt.executeUpdate(sql);

if(i>0)

System.out.println("Row inserted");

else

System.out.println("Unable to insert");

con.close();

}

catch(Exception e){ System.out.println(e);}

}

}

**#5.Table Updation by accepting input from the user**

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.Statement;

import java.util.\*;

class MysqlConnection

{

public static void main(String args[])

{

Scanner sc = new Scanner(System.in);

try

{

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

Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/vsem","root","root");

System.out.println("Enter Roll No to update the name:");

String a=sc.next();

System.out.println("Enter Name:");

String b = sc.next();

Statement stmt=con.createStatement();

String sql="update std set name ='"+b+"' where rno ='"+a+"'";

int i = stmt.executeUpdate(sql);

if(i>0)

System.out.println("Row updated");

else

System.out.println("Unable to update");

con.close();

}

catch(Exception e){ System.out.println(e);}

}

}

**#6.Deleting a row from the table**

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.Statement;

import java.util.\*;

class MysqlConnection

{

public static void main(String args[])

{

Scanner sc = new Scanner(System.in);

try

{

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

Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/vsem","root","root");

System.out.println("Enter Roll No to delete the rcord");

String a=sc.next();

Statement stmt=con.createStatement();

String sql="delete from std where rno ='"+a+"'";

int i = stmt.executeUpdate(sql);

if(i>0)

System.out.println("Row deleted");

else

System.out.println("Unable to delete");

con.close();

}

catch(Exception e){ System.out.println(e);}

}

}

**#7.Insertion Using PreparedStatement**

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.Statement;

import java.util.\*;

class MysqlConnection

{

public static void main(String args[])

{

Scanner sc = new Scanner(System.in);

try

{

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

Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/cseab","root","root");

System.out.println("Enter Roll No:");

String a=sc.next();

System.out.println("Enter Name:");

String b = sc.next();

PreparedStatement ps=con.prepareStatement("insert into std values(?,?)");

ps.setString(1,a);

ps.setString(2,b);

int i=ps.executeUpdate();

if(i>0)

System.out.println("Row inserted");

else

System.out.println("Unable to insert");

con.close();

}

catch(Exception e){ System.out.println(e);}

}

}

**#8.Updation Using PreparedStatement**

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.Statement;

import java.util.\*;

class MysqlConnection

{

public static void main(String args[])

{

Scanner sc = new Scanner(System.in);

try

{

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

Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/cseab","root","root");

System.out.println("Enter Roll No to update the name:");

String a=sc.next();

System.out.println("Enter Name:");

String b = sc.next();

PreparedStatement ps=con.prepareStatement("update std set name=? where rno=?");

ps.setString(1,b);

ps.setString(2,a);

int i=ps.executeUpdate();

if(i>0)

System.out.println("Row updated");

else

System.out.println("Unable to update");

con.close();

}

catch(Exception e){ System.out.println(e);}

}

}

**#9.Fetch the data from table std**

import java.sql.\*;

class MysqlCon{

public static void main(String args[]){

try{

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

Connection con=DriverManager.getConnection(

"jdbc:mysql://localhost:3306/vsem","root","root");

//here sonoo is database name, root is username and password

Statement stmt=con.createStatement();

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

while(rs.next())

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

con.close();

}catch(Exception e){ System.out.println(e);}

}

}