UNIT - I

CHAPTER 2

Java Database Connectivity (JDBC)

Syllabus Topics

JDBC : Introduction, JDBC Architecture, Types of Drivers, Statement, ResultSet, Read Only ResultSet, Updatable ResultSet, Forward Only ResultSet, Scrollable ResultSet, PreparedStatement, Connection Modes, SavePoint, Batch Updations, CallableStatement, BLOB & CLOB

Syllabus Topic : JDBC- Introduction

2.1 Introduction

Q. Explain the role of JDBC in JAVA.
Q. What is JDBC ? What it does ?

- **JDBC :** JDBC (Java Database Connectivity) is Java based data access technology and used for Java database connectivity.
- It is an Application Programming Interface (API) used to connect Java application with various Database instance.
- Database can be Oracle, MSAccess, MySQL and SQL Server.
- JDBC allows java program to execute SQL statement (query) and retrieve result from database.
- The JDBC library includes APIs for following tasks that are commonly associated with database usage.
  1. Establish a connection to a database
  2. Issue SQL statements
  3. Process the results

- Fig. 2.1.1 illustrates working of JDBC model.

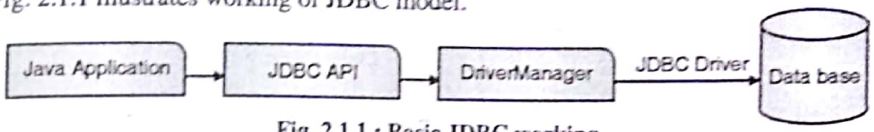

Fig. 2.1.1 : Basic JDBC working

- JDBC allows user to access any tabular data sources. For example: spreadsheets, relational databases, or even flat files. User writes SQL query to java methods in the JDBC classes and gets JDBC objects that represent the results of our query. JDBC also allow us to access an ODBC-based database using a JDBC-ODBC bridge. JDBC is portable since Java is portable across platforms.

Syllabus Topic : JDBC Architecture

2.2 JDBC Architecture

Q. Write a short note on JDBC Architecture.
Q. Explain the two-tier and three-tier models of JDBC.

- JDBC architecture allows a java program to access the database table for data storage and retrieval. Database access can be done by both two-tier and three-tier architecture model.
- JDBC Architecture consists of :

(a) **JDBC API :** Consists of set of classes and interfaces that support java application to access the permanent storage. The JDBC API allows connecting to various databases using respective drivers and DriverManager interface.

(b) **JDBC Driver API :** Driver API support connection to a particular database for data access.

A. Two-tier Architecture for database Access

- In the two-tier JDBC model, Java applet or application communicates directly to the data base. This model needs a specific JDBC driver that can connect and communicate with a database system.
- SQL command from the client application is send to the database or other data source.
- Once the SQL statement is executed, the result is forwarded back to the client machine.
- The data source can be located in a network environment to a particular database server. Thus data can be accessed between client and server through client/server communication.
- The network can be an intranet; for example, connects all the department of a company or it can be the Internet.

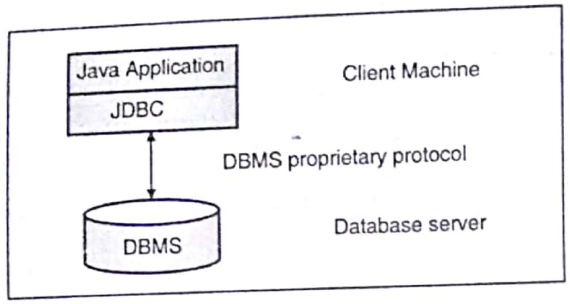

Fig. 2.2.1 : Two-tier Architecture

B. Three-tier Architecture for database Access

- In the three-tier model, SQL commands are forwarded from the client machine to a middle- tier server first. Middle-tier server forwards the commands to the data base server. The data base server retrieves the incoming SQL commands, execute the command and send the result back to the middle-tier server. Middle-tier server then forwards it to the client machine.
- Three-tier model is more efficient than two-tier model. Data can be processed at the middle tier before sending to the database server during data storage. During data retrieval also data can be processed before forwarding it to the client machine.
- This model improves the deployment of application architecture; hence improve the performance of the application.

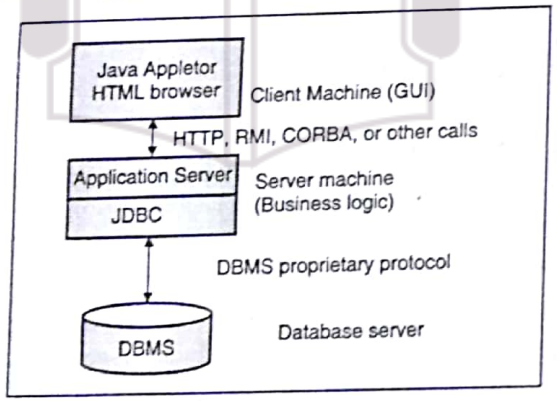

Fig. 2.2.2 : Three-tier Architecture

- *Middle tier can be developed in languages like C or C++ which makes the processing of middle-tier more faster.*
- *Optimizing compilers for java can be used for the middle-tier development with java programming features like multithreading and enhanced security.*

Syllabus Topic : Types of Drivers

2.3 Types of JDBC Driver

Q. Describe the different types of JDBC Driver.

Q. Explain the steps to be followed in Java Database Connectivity for accessing database table content.

- **JDBC :** JDBC driver for a particular database system need to be loaded by a Java program before establishing a connection to the database. Class.forName() method can be used to load the JDBC driver as :

```
Class.forName("driver-name");
```

- **For example :** To connect with mysql database using native JDBC driver; the command is

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

- The JDBC Driver Manager class sends all database related calls and execution statements to the loaded driver.
- There are different types of JDBC Drivers as

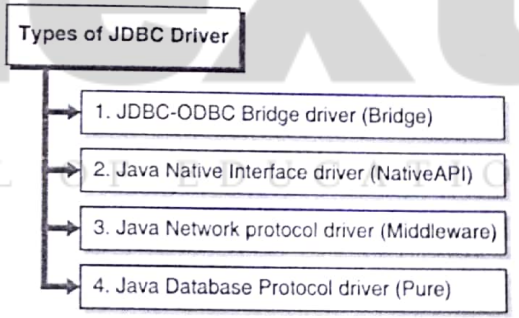

Fig. C2.1 : Types of JDBC Driver

➔ Type 1 : JDBC-ODBC Bridge driver

- The Type-1 driver or JDBC-ODBC bridge driver converts the statements written in Java Database Connectivity (JDBC) form into Open Database Connectivity (ODBC) statements and send it to the ODBC driver.
- The JDBC-ODBC Bridge driver is generally used for trail use not for the implementation of application development.
- To create JDBC connection to the database, ***java.sql*** package should import into the program.

- The steps to be followed in JDBC to access a database table are

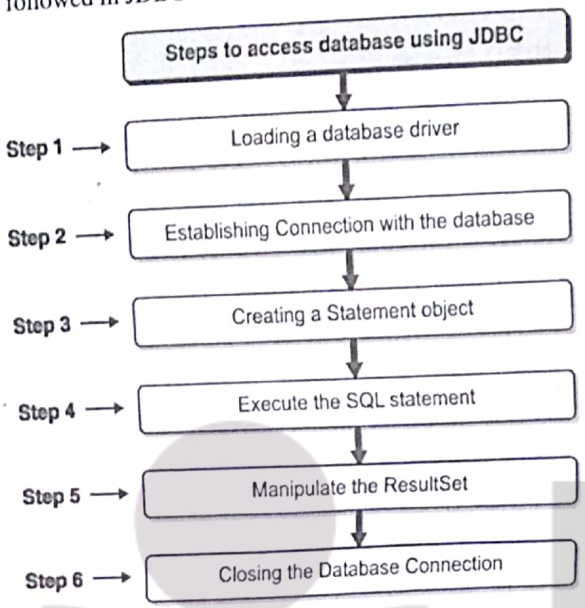

Fig. C2.2 : Steps to access database using JDBC

Step 1 : Loading a database driver

- Database driver can be loaded by calling Class.forName() method with driver-name as the argument.
- Once loaded, the Driver class creates an instance of itself.
- Client can connect to a database server through JDBC Driver.
- **For example :** JDBC-ODBC Bridge driver can be loaded as

```
try
{
   Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
}
catch(Exception e){System.out.println("Error1:"+e.getMessage());}
```

- **For example :** Mysql native api driver can be loaded as

```
try
{
   Class.forName("com.mysql.jdbc.Driver");
}
catch(Exception e){System.out.println("Error1:"+e.getMessage());}
```


Step 2 : Establishing Connection with the database

- Driver Manager.getConnection() method is used to establish connection with the database as

```
Connection con = DriverManager.getConnection(url, "loginname", "password")
```

Where,

(i) url - URL of the database

(ii) loginname - user name of the database

(iii) password - password of the database

- **For example :** Statement required to establish connection with a MySql database named "db" with user name and password as "root" and "shajilpa"

```
Connection con= DriverManager.getConnection(" jdbc:mysql://localhost/db", "root",
"shajilpa");
```

Step 3 : Creating a Statement object

- A statement objects is used to send and execute SQL statements.
- Three kinds of Statements are :

Sr. No.	Interface	Use
1.	Statement	Used to execute static SQL statements. It cannot accept parameters.
2.	PreparedStatement	Used to execute dynamic SQL statements. It accepts input parameters at runtime.
3.	CallableStatement	Used to call database stored procedures. Callable Statement interface accepts input parameters at runtime.

- Established Connection can be used to interact with the database, by creating a Statement object using createStatement() method.
- **For example :** Format of creating a Statement object from the Connection named "con" is

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

- Prepared Statement object can be created from the connection object using prepareStatement() method by specifying the sql query
- **For example :** Format of creating a Prepared Statement object from the Connection named "con" is

```
String sql1="insert into dept table values(?,?,?)";
PreparedStatement ps=con.prepare statement(sql1);
```

- CallableStatement object can be created from the connection object using prepareCall( ) method by specifying the sql procedure call query.

- **For example :** format of creating a CallableStatement object from the Connection named "con"

```
String sql="{calladdData(?,?,?)}";
CallableStatement st=con.prepareCall(sql);
```

Step 4 : Execute the SQL statement

- Statement interface defines methods to execute the SQL queries.
- executeQuery(), executeUpdate() and execute() are the methods of Statement object to execute SQL queries.

Sr. No.	Methods	Use
1.	executeQuery()	Used to execute a SELECT statement, which returns the ResultSet as records
2.	executeUpdate()	Used to execute data definition language statements that create or modify table and data manipulation statements for its content
3.	execute()	Used to execute an SQL statement written as String object.

Step 5 : Manipulate the ResultSet

- **ResultSet :** The ResultSet stores table of data, that is generated by executing SQL select statements.
- **Cursor :** A resultSet maintains a cursor point to the current row.
- **next() method :** It is used to move forward through the rows of the resultset.
- **ResultSetMetaData :** It is used to fetch the meta-data about the table like type and property of the columns of a database table. It is created from a ResultSet object.
- **For example :** Code to display the entire content of dept table with three attributes(deptno, deptname, deptloc)

```
String sql="select * from dept table";
ResultSet rs=st.executeQuery(sql);
while(rs.next())
{
    System.out.print(rs.getInt(1)+"\t");
    System.out.print(rs.getString(2)+"\t");
    System.out.print(rs.getString(3)+"\n");
}
```

Step 6 : Closing the Database Connection

- Connection.close() method can be used to close the database connection and release all the resources used.

- **For example :** Format for closing the Connection named"con"is :

```
con.close();
```

- The format of data flow is in JDBC-ODBC Bridge driver is : Client->JDBCDriver->ODBCDriver->Database.

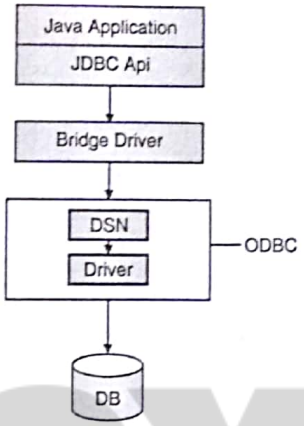

Fig. 2.3.1: JDBC-ODBC Bridge driver

- The driver is implemented in the "sun.jdbc.odbc.JdbcOdbc Driver" class.

Advantage

Many Relational Database Management Systems (RDBMS) supports the ODBC call. Thus JDBC-ODBC Bridge driver can be used easily.

Disadvantages

- **Not Portable :** Bridge driver is not written fully in Java; it is Microsoft platform dependent.
- **Slowest of All driver types :** For each database access statement; JDBC calls need to convert into ODBC calls and vice versa that makes the driver to slow down the process.
- ODBC needs **to be installed in the client system.**
- **Cannot used for internet** based web applications.
- **Applets cannot load** native code, thus bridge driver is not useful for applet programs.

➜ **Type 2 : Java Native Interface Driver (Native API)**

- Type-2 driver or Native-API driver converts the JDBC statements into native database specific statements.

  i.e. Type-2 driver is specific to a particular database. For e.g. Oracle will have oracle native API driver, mysql will have mysql native api driver etc.

- The format of dataflow in Native API driver is :

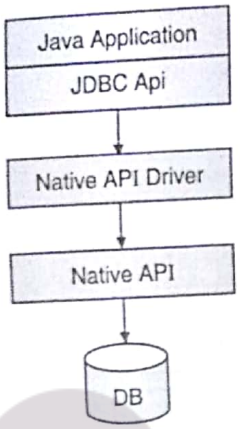

Fig. 2.3.2 : Native API

☞ **Advantage**

A type-2 driver has better performance than bridge drivers as jdbc to odbc translation is not needed.

☞ **Disadvantages**

- If the database for an application changes native API driver need to be changed.
- Not completely portable as all Type-2 drivers may not written in Java Language.
- Native API driver need to be in the Client System, so cannot be used with the Internet applications.
- Type-2 driver cannot be used with an applet as it cannot load native code.

➜ **Type 3 : Java Network protocol driver (Middleware)**

- The Type-3 driver or network-protocol driver use a middle-tier between the program and the database.
- Type-3 driver translates JDBC statements into net protocol format which is then translated into a DBMS protocol by middle-tier server.
- The middle-tier server can be Type-1, Type-2 or Type-4 drivers.

- The format of data flow in Network protocol or Middleware driver is :

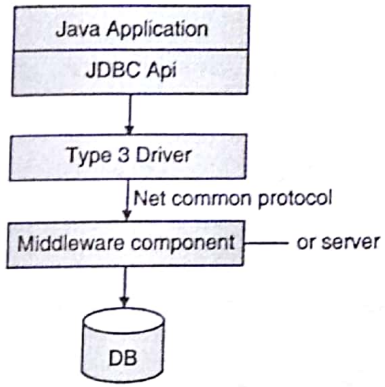

Fig. 2.3.3 : Middleware

☞ **Advantages**

- Portable and platform independent as it completely written in Java; hence used or the web applications.
- It supports middleware services such as caching, load balancing, and advanced system administration such as logging and auditing.
- It allows to access multiple databases as it act as an interface.
- For Network-protocol driver, client JDBCdriver is small and fast to load.

☞ **Disadvantages**

- It requires middleware server application.
- Traversing the recordset at the client takes longer time, as the data comes through the Middleware.
- Requires database specific coding to be done in the middle tier.

➜ **Type 4 : Java Database Protocol driver (Pure)**

- Type-4 or Database protocol driver converts JDBC statements directly to the specific database statements. It uses java networking libraries to access the database server.
- It is installed inside the Java Virtual Machine of the client.

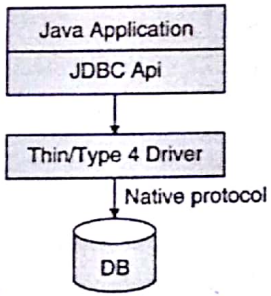

Fig. 2.3.4 : Pure

☞ **Advantages**

- Type-4 drivers are completely written in Java to provide platform independence for web applications.
- Provides better performance over other drivers as no translation is required.

☞ **Disadvantage**

At client side, a separate driver is needed for each database.

2.4 JDBC Driver Manager

Q. Explain the role of DriverManager class in JDBC. Explain any six methods available in this class.

Q. Write a short note on Driver Manager class.

- The Driver Manager is an intermediate between the user program and JDBC drivers in a java application.
- Driver Manager maintains the available database drivers for an application.
- It can be used to establish the connection between a database and the database driver.
- Driver Manager.registerDriver() method can be used to maintain the driver required for a program.
- Driver Manager can be used to load the driver from the "jdbc.drivers" system property.
- JDBC driver loaded using Class.forName() method can be used by DriverManager to establish the connection.

☞ **Methods of Driver Manager class**

(i) public static void registerDriver(Driver driver)

It is used to register a database driver with Driver Manager.

(ii) public static void deregisterDriver(Driver driver)

It is used to remove the registration done by a database driver.

(iii) public static Connection getConnection(String url)

It is used to establish a connection with the specified url, indicating the database.

(iv) public static Connection getConnection(String url,String userName,String password)

It is used to establish a connection with a database using the url, username and password of a specific database.

For example : Statement required to establish connection with a MySql database named "db" with user name and password as "root" and "shajilpa"


```
Connection con = DriverManager.getConnection( "jdbc:mysql://localhost/db", "root",
"shajilpa");
```

(v) public static DrivergetDriver(String url)

It is used to locate a driver with a URL.

(vi) public static Enumeration<Driver>getDrivers()

It retrieves all the currently loaded JDBC drivers

(vii) public static void setLoginTimeout(int seconds)

It is used to set the maximum time period in seconds while establishing a connection with the database.

(viii) public static int getLoginTimeout()

It is used to get the maximum time period in seconds while establishing a connection with the database.

2.5 JDBC Connection Interface

Q. Explain Connection interface along with its methods.

Q. Discuss the role of Connection interface in JDBC.

- Connection interface represent the database connection between a database and a JDBC driver in a java application.
- DriverManager.getConnection() method can be used to establish a connection with the database as

```
Connection con = DriverManager.getConnection( url, "loginname", "password")
```

Where,

1. Url – Specifies the URL of the database

2. loginname – Username of the database

3. password – Password of the database

- **For example :** statement required to establish connection with a MySql database named "db" with username and password as "root" and "shajilpa"

```
Connection con = DriverManager.getConnection( "jdbc:mysql://localhost/db", "root",
"shajilpa");
```

- Three overloaded DriverManager.getConnection() methods are :
  (a) getConnection(String url)
  (b) getConnection(String url,Properties prop)
  (c) getConnection(String url,String user,String password)

– A database URL is an address that points to the database. Important JDBC driver names and database URL are :

RDBMS	JDBC driver name	Database URL
MySQL	com.mysql.jdbc.Driver	jdbc:mysql://hostname/databaseName
ORACLE	oracle.jdbc.driver.OracleDriver	jdbc:oracle:thin:@hostname:portNumber:databaseName

☞ **Connection interface methods**

(a) **public void abort(Executor executor)**

It is used to terminate an open connection.

(b) **public void close()**

It is used to close and existing connection object. It releases all the database resources associated with the connection.

(c) **public void commit()**

It commits all the previously executed SQL commands permanently.

(d) **public void rollback()**

It discards all the changes made during the execution of previous SQL commands and reverts back to the state where previous commit was done.

(e) **public void setAutoCommit(boolean autoCommit)**

It is used to set the auto-commit mode as either true or false.

(f) **public boolean getAutoCommit()**

It is used to fetch the auto-commit mode.

(g) **public boolean isClosed()**

It is used to find whether the Connection object is closed or not.

(h) **public booleanisReadOnly()**

It is used to find whether the Connection object is in read-only mode.

(i) **public Statement createStatement()**

It is used to create a Statement object for executing the static SQL queries.

(j) **public Statement createStatement(int resultSetType, int resultSetConcurrency)**

It is used to create a Statement object with the specified result set and concurrency type.

(k) **public PreparedStatement prepareStatement(String sql)**

It is used to create a PreparedStatement object for executing dynamic SQL queries.

(l) **public PreparedStatement prepareStatement(String sql, int resultSetType, int resultSetConcurrency)**

It is used to create a PreparedStatement object with the specified result set and concurrency type.

(m) **public CallableStatement prepareCall(String sql)**

It is used to create a CallableStatement object for calling a stored procedure.

(n) **public CallableStatement prepareCall(String sql, int resultSetType, int resultSetConcurrency)**

It is used to create a CallableStatement object with the specified result set and concurrency type.

(o) **public DatabaseMetaDatagetMetaData()**

It is used to create a Database MetaData object to retrieve the meta data about the database.

(p) **public int getNetworkTimeout()**

It is used to retrieve time in milliseconds the driver will waitfor the executing of a query.

Syllabus Topic : Statement

2.6 JDBC Statement Interface

Q. Explain Statement interface along with its methods.

Q. Discuss the role of Statement interface in JDBC.

– CallableStatement, and PreparedStatement interfaces has the methods and properties that enable us to pass SQL or PL/SQL commands and get data from our database.

– It also has methods that help bridge data type dissimilarities between Java and SQL data types used in a database.

☞ **JDBC Statements Interface**

– The Statement interface can be used to execute static SQL queries with the database.

– Maximum one ResultSet object can be open for a Statement object at a time.

– If more than one ResultSet object is required to form the result, then each one must be created by different Statement objects.

– If the ResultSet object is kept open then none of the execution methods in the Statement interface works

– createStatement() method of Connection can be used to create a Statement object.

– **For example:** Format of creating a Statement object from the Connection named"con"is :

```
Statementst=con.createStatement();
```

☞ **Methods of Statement interface**

(a) public ResultSet executeQuery(String sql)

It is used to execute data retrieval SQL SELECT query.

(b) public int execute Update(String sql)

It is used to execute data manipulation and definition query like create, drop, insert, update, delete etc.

(c) public boolean execute(String sql)

It is used to execute queries that will return multiple results.

(d) public int[] executeBatch()

It is used to execute batch of SQL commands together.

(e) public void add Batch(String sql)

It is used to add an SQL command to the batch of commands

(f) public void clearBatch()

It is used to clear the batch-command list

(g) public voidclose()

It is used to close a statement object

(h) public void closeOnCompletion()

It is used to close a statement object once all its associated resultsets are closed.

(i) public void setQueryTimeout(int seconds) and Public int getQueryTimeout()

It is used to set and get time in seconds for the statement to execute the query.

Program 2.6.1 : *Write a java program to insert one record of information into the database table named (dept table), by accepting the details from the user; using Statement object.*

Hint : *Table Detail - depttable(deptno int primary key, deptname varchar(10), deptloc varchar(10))*

Soln. :

MYSQL Command

```
create database db;
use db;
create table depttable(deptno int primary key, deptname varchar(10), deptloc varchar(10));
```

Java Program

```
import java.sql.*;
import java.io.*;
class deptinsert
{
   Public static void main(String args[])throws SQLException
   {
         try
         {
              Class.forName("com.mysql.jdbc.Driver");
              Connection con =
DriverManager.getConnection("jdbc:mysql://localhost/db","root","shajilpa");

              Statement st=con.createStatement();
              BufferedReader br=newBufferedReader(new InputStreamReader(System.in));
              System.out.print("EnterDept-No:");
              int no=integer.parseInt(br.readLine());
              System.out.print("EnterDept-Name:");
              String name=br.readLine();
              System.out.print("EnterDept-Location:");
              String loc=br.readLine();
              String sql1="insert into depttable values("+no+",'"+name+"','"+loc+"')";
              st.executeUpdate(sql1);
              System.out.println("\n\t\tRecordSaved...");
              con.close();
         }
         catch(Exception e){System.out.println("Error1:"+e.getMessage());}
   }
}
```

Syllabus Topic : JDBC ResultSet : Read Only, Updatable, Scrollable, Forward Only

2.7 JDBC ResultSet : Read Only, Updatable, Scrollable, Forward only

Q. Explain various ways of creating ResultSet in JDBC.

Q. What is ResultSet Downgrade rule in JDBC?

Q. Discuss various methods of ResultSet interface.

- The **java.sql.ResultSet** interface represents the result set of a database query. SQ statement that reads data from database, always stores it in Result set.
- The object of ResultSet maintains a **cursor** pointing to a row of a table. In the beginni cursor points to before the first row. In default case, ResultSet object is not updatable a it can be moved forward only.

A. ResultSet creation

- The method of Connection class provides the format of mentioning the result set ty and a concurrency type as :
  (a) Statement createStatement(int resultSetType, int resultSetConcurrency)
  (b) PreparedStatement prepareStatement (String sql, int resultSetType, int resultSetConcurrency)
  (c) CallableStatement prepareCall (String sql, int resultSetType, int resultSetConcurrency)
- **Static constant values for result set type are :**
  (i) ResultSet.TYPE_FORWARD_ONLY
  (ii) ResultSet.TYPE_SCROLL_INSENSITIVE
  (iii) ResultSet.TYPE_SCROLL_SENSITIVE
- **Static constant values for concurrency type are :**
  (i) ResultSet.CONCUR_READ_ONLY
  (ii) ResultSet.CONCUR_UPDATABLE
- Once Statement, PreparedStatement, or CallableStatement object, has been created it properties can be retrieved as :
  (i) int getResultSetType() throws SQLException - returns the result set type.
  (ii) int getResultSetConcurrency() throws SQL Exception - returns the concurrency type.

B. Result Set Downgrade Rules

- If the resultset type or concurrency type set is not suitable for the JDBC driver then i takes the other types with the downgrade rules as :

  - If result set type is TYPE_SCROLL_SENSITIVE, and the JDBC driver cannot permit, then it changes to TYPE_SCROLL_INSENSITIVE.
  - If result set type is TYPE_SCROLL_INSENSITIVE, and the JDBC driver cannot permit, then it changes to TYPE_FORWARD_ONLY.
  - If concurrency type is CONCUR_UPDATABLE, and the JDBC driver cannot permit, then it changes to CONCUR_READ_ONLY.

C. Result Set Methods

Result Set Methods for Scrollable result set

(i) void beforeFirst() throws SQLException
Cursor position before the first row

(ii) void afterLast()throwsSQLException
Cursor position aftert he lastrow

(iii) boolean first() throws SQLException
Cursor move to first row

(iv) boolean last() throws SQLException
Cursor move to last row

(v) boolean absolute(int row) throws SQLException
1. Cursor positions to an absolute position.
2. If row value is positive - move from beginning
3. If row value is negative - move from end

(vi) boolean relative(int row) throws SQLException
1. Cursor move relative to the current row.
2. If row is positive move forward
3. If row is negative move backward

☞ **ResultSet Methods for Checking the Current Position in a scrollable result set**

(a) boolean isBeforeFirst() throws SQL Exception

It return true if cursor is before first row

(b) boolean isAfterLast() throws SQLException

It returnt rue if cursor is after last row

(c) boolean isFirst() throws SQLException

It return true if cursor is at first row.

(d) **boolean isLast() throws SQLException**

It return true if cursor is at last row

(e) **int getRow() throws SQLException**

It return the current row number

(f) **boolean next() throws SQLException and boolean previous() throws SQLException**

These methods are used for move the cursor in forward and backward direction.

For example : To display the details (empno, salary) from the emp table in the reverse order.

```
Statement stmt=con.createStatement
(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);
ResultSet rs=stmt.executeQuery("SELECT empno,sal FROM emp");
rs.afterLast();
while(rs.previous())
{
        System.out.println(rs.getString("empno")+""+rs.getFloat("sal"));
}
```

Syllabus Topic : PreparedStatement

2.8 JDBC PreparedStatement Interface

Q. Explain PreparedStatment with an example.

Q. Write a short note on 'PreparedStatement'. Explain with the code specification

☞ **PreparedStatement interface**

- PreparedStatement interface is used to execute dynamic SQL queries. It is created with a specific SQL command with place holders that can be executed multiple times by assigning dynamic values for the place holder.
- Place holder is represented by a meta character(?)

A. Creating a Prepared Statement : Connection.prepareStatement() method is used to create a prepared statement object.

☞ **For example**

```
String sql="select*from people where id=?";
PreparedStatement ps=connection.prepareStatement(sql);
```


B. Inserting Parameters into a PreparedStatement : To insert a parameter into the SQL query, specify a question mark (?).

☞ **For example**

```
String sql="select*from people where id=?";
```

insert parameters at the location of the question mark using set___()method.

☞ **For example**

```
ps.setLong(1,123);
```

where,

- 1 is the index of the parameter(?)
- 123 is the value to insert into the SQL statement.

More parameter can be represented by multiple question marks.

☞ **Example**

```
String sql="select * from people where firstname=? and lastname=?";
PreparedStatement ps=connection.prepareStatement(sql);
ps.setString(1,"Shajil");
ps.setString(2,"Kumar");
```

C. Executing the PreparedStatement : Use either executeQuery() or executeUpdate() method to execute the prepared statement query.

☞ **For example**

```
ResultSet result=ps.executeQuery();
```

- executeQuery() method returns a ResultSet that store all the records which are fetched
- The executeUpdate() method is used when updating the database.

☞ **Example**

```
String sql="update people set firstname=?,lastname=?where id=?";
PreparedStatement ps=connection.prepareStatement(sql);
ps.setString(1,"Shajil");
ps.setString(2,"Kumar");
ps.setLong(3,123);
int rowsAffected=ps.executeUpdate();
```

D. Reusing a PreparedStatement : It can be used to execute the query multiple times by changing the parameter values.

☞ **Example**

```
String sql="update people set firstname=?,lastname=?whereid=?";
PreparedStatement ps=connection.prepareStatement(sql);
ps.setString(1,"Shajil");
ps.setString(2,"Kumar");
ps.setLong(3,123);
int rowsAffected=ps.executeUpdate();

ps.setString(1,"Amrutha");
ps.setString(2,"Nair");
ps.setLong(3,456);
int rowsAffected=ps.executeUpdate();
```

☞ **Methods of Prepared Statement**

(a) public boolean execute()

It is used to execute the SQL statement.

(b) public ResultSet executeQuery()

Used to execute the SQL SELECT query in the Prepared Statement and return the ResultSet object that fetches the records.

(c) public int executeUpdate()

It is used to execute the SQL Data Manipulation Language (DML) queries like insert, update or delete.

(d) public void clear Parameters()

It is used to clear all the parameter values for resetting it later.

(e) public void setBoolean(int parameterIndex,boolean x)

(i) public void setByte(int parameterIndex, byte x)

(ii) public void setDouble(int parameterIndex, double x)

(iii) public void setFloat(int parameterIndex, float x)

(iv) public void setInt(int parameterIndex, int x) etc.

These methods are used to set the parameter value by specifying the parameter index and its value.

Program 2.8.1 : Write a java program to insert one record of information into the database table named (depttable),by accepting the details from the user; using PreparedStatement.

Soln. :

Table Detail

```
depttable(deptno int primary key, deptname varchar(10), deptloc varchar(10))
```

MYSQL Command

```
Create database db;
Use db;
Create table depttable(deptno int primary key,deptname varchar(10),deptloc varchar(10));
```

```
import java.sql.*;
import java.io.*;
class deptinsertps
{
   public static void main(String args[]) throws SQLException
   {
   try
      {
         Class.forName("com.mysql.jdbc.Driver");
         Connection
   con=DriverManager.getConnection("jdbc:mysql://localhost/db","root","shajilpa");

         String sql1="insert into depttable values(?,?,?)";
         PreparedStatement ps=con.prepareStatement(sql1);

         BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

         System.out.print("Enter Dept-No : ");
         int no = Integer.parseInt(br.readLine());
         System.out.print("Enter Dept-Name : ");
         String name = br.readLine();
         System.out.print("Enter Dept-Location : ");
         String loc = br.readLine();
         ps.setInt(1, no);
         ps.setString(2,name);
         ps.setString(3,loc);

         ps.executeUpdate();
```

```
        System.out.println("\n\t\tRecord Saved...");

        con.close();
    }
  catch(Exception e){ System.out.println("Error1:"+e.getMessage()); }
  }
}
```

Syllabus Topic : CallableStatement

2.9 JDBC CallableStatement Interface

Q. Explain CallableStatement.

Q. Write a short note on 'CallableStatement'. Explain with the code specification.

- CallableStatement is used to call a database stored procedure. A CallableStatement c return one or more ResultSet after executing the procedure.
- Three types of parameters used with the stored procedure are : IN, OUT, and INOU defined as

Sr. No.	Parameter	Description
1.	IN	IN parameter value is unknown initially while sql statement is written. IN parameter values can be assigned using set__() method.
2.	OUT	OUT parameter value can be returned after the execution of stored procedure. It can be retrieved using the get___() method.
3.	INOUT	It is a parameter that can be used for both input and output values. Value can be assigned using set___() method and retrieved using get___() method.

A. Creating a Stored Procedure

Consider the stored procedure get StuName for STU database that take stu_id a parameter and returns stu_firstname.

```
DELIMITER$$
CREATE PROCEDURE`STU`.`getStuName`
(IN STU_ID INT,OUT STU_FIRST VARCHAR(25))
BEGIN
SELECT first INTO STU_FIRST
FROM student
WHERE ID= STU_ID;
```



```
END$$
DELIMITER;
```

B. Creating CallableStatement Object

Connection.prepareCall() method can be used to create a CallableStatement by specifying the procedure call as parameter.

For example : Let con is a Connection object.

```
CallableStatement cs=null;
try{
      String SQL="{callgetEmpName(?,?)}";
      cs=con.prepareCall(SQL);
      ...
  }
catch(SQLException e){
...
}
```

Where,

- Variable SQL stores the stored procedure call with parameter place holders.
- IN parameter value can be assigned using set____() method
- Once stored procedure is executed, OUT parameter value can be retrieved using get___() method.

C. Setting Parameter Values

- Created CallableStatement parameters(?) can be changed using set___() method

☞ **For example :** To calculate Salary() method by passing the employee name and basic pay.

```
CallableStatement cs=connection.prepareCall("{callcalculateSalary(?,?)}");
cs.setString(1, "xyz");
cs.setInt  (2, 10000);
```

D. Executing the CallableStatement

Execute Query() or executeUpdate() method can be used to execute the CallableStatement

executeQuery() method returns a ResultSet after the procedure call.

☞ **For example**

```
ResultSet result=cs.executeQuery();
```

executeUpdate() method updates the database with the procedure call.

☞ **For example**

```
cs.executeUpdate();
```

E. Closing CallableStatement Object

close() method closes a CallableStatement object.

Program 2.9.1 : Write a java program to insert one record of information into the database table named (depttable), by accepting the details from the user; using CallableStatement.

Soln. :

Table Detail: depttable(deptno int primary key, deptname varchar(10),deptloc varchar(10))

MYSQL Command

- create database db;
- use db;
- create table dept table(deptno int primary key,deptname varchar(10),deptloc varchar(10));

MYSQL Command to create the stored procedure

```
delimiter$$
create procedure addData(inno int,inname varchar(10),inloc varchar(10))
begin
insert into dept table(deptno,deptname,deptloc) values(no,name,loc);
end$$
```

Deptinsertcs.java

```
Import java.sql.*;
Import java.io.*;
Class dept insert cs
{
  Public static void main(String args[]) throws SQLException
  {
      try
      {
          Class.forName("com.mysql.jdbc.Driver");
          Connection
con=DriverManager.getConnection("jdbc:mysql://localhost/db","root","shajilpa");

          CallableStatement cs=con.prepareCall("{call addData(?,?,?)}");

          BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
```



```
          System.out.print("Enter Dept-No:");
          Int no=Integer.parseInt(br.readLine());
          System.out.print("EnterDept-Name:");
          String name=br.readLine();
          System.out.print("EnterDept-Location:");
          String loc=br.readLine();
          cs.setInt(1,no);
          cs.setString(2,name);
          cs.setString(3,loc);
          cs.executeUpdate();
          System.out.println("\n\t\tRecordSaved...");
          con.close();
      }
  catch(Exception e){System.out.println("Error1:"+e.getMessage());}
  }
}
```

MYSQLCommand to view the records

```
select * from dept table;
$$
```

Syllabus Topic : Connection Modes, SavePoints, Batch Updations

2.10 JDBC Connection Modes, SavePoint, Batch Updations

- JDBC Connection has auto-commit mode. If it is true (which it is by default), then on completion every SQL statement is **committed** to the database. Transactions enable you to control whenever changes are applied to the database.
- Here a single SQL statement or a group of SQL statements are treated as one logical unit, and if even single statement fails, the whole transaction fails.
- We use the Connection object's **setAutoCommit()** method to enable manual- transaction support for JDBC driver.

```
conn.setAutoCommit(false);
```

In JDBC, **Connection interface methods** can be used to manage the transaction as

(a) public void setAutoCommit(boolean status)

If auto-commit is assigned as true, each individual statement is committed by default.

(b) public void commit()

It is used to commit the whole transaction.

(c) public void rollback()

It is used to cancel the entire transaction.

A. Committing Transactions

- SQL statements won't get committed if auto-commit mode is disabled. Initially disable the auto commit, execute set of statements part of a single transaction and last call the commit() method as

```
try{
        con.setAutoCommit(false);          //MultiplesetofSQLstatementsexecution.
            con.commit();
   }
}catch(SQLException e)
{
   con.rollback();
}
```

- Commit() method make the transaction changes permanent.

B. Using Transactions to Preserve Data Integrity

When multiple users access the shared data transactions, the data integrity can be gained using the methods :

- getTransactionIsolation() : It is used to find the transaction isolation level.
- SetTransactionIsolation() : It is used to change the isolation level for the Connection.

C. Setting and Rolling Back to Savepoints

- Savepoint can be used to indicate the starting of a transaction, so that it can rollback to the savepoint during the occurrence of an error.
- Connection.setSavepoint() method set a savepoint object.
- Connection.rollback() method can specify the name of the save point where in which it has to rollback.

☞ **Example :** To set savepoint as

```
Savepointsave1=con.setSavepoint();
```

Rolls back at Savepoint as :

```
con.rollback(save1);
```


D. Releasing Savepoints

- Connection.releaseSavepoint()can be used to clear a savepoint which is already set.
- Once the transaction is committed all the savepoints become invalid.

E. When to call rollback Method

- Connection.rollback() method clear the entire transaction effects and return to the state before the starting of the transaction. Thus every occurrence of an exception cannot be processed by calling the rollback() method.

2.10.1 SavePoint

- The save point is a logical position in a transaction which is used as marker. This is a marker up to which we can rollback the transaction.
- When the save point is placed in the middle of the transaction, the logics placed before the save point will be **committed** and the logics placed after the save point will be **rolled back**.
- There are **two types of save points** :
  1. Named save point
  2. Unnamed save point
- Save points are available in java.sql.Savepoint interface.
- Save point placed in the transaction are reference it using connection.rollback () not using connection.comit(). That means we cannot commit up to save point position, but we can roll back up to save point position.

Institute table: (IName varchar(20),IAddress varchar(20),ICode number(15))

SavepointDemo.java

```
import java.sql.*;
public class SavepointDemo
{
   public static void main (String args []) throws Exception
   {
        Class.forName ("oracle.jdbc.driver.oracleDriver");
        Connection con =
        DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl", "scott",
        "tiger");
        Statement stmt = con.createStatement ();
        con.setAutoCommit (false);
        stmt.executeUpdate ("insert into Institute values ('Tech_MAX', 'mumbai', 1978)");
        Savepoint sp = con.setSavepoint ("ABC");
        st.executeUpdate ("update Institute set Iname= 'BPO' where Icode=120");
```

```
        st.executeUpdate ("insert into Institute values ('Tech_Max_2', 'Pune', 812)");
        con.rollback(sp);
        con.commit ();
    }
}
```

- Savepoint sp=con.setSavepoint ("ABC"): After execution of some database operation the save point is created at this position, and any name can be passed as string.
- con.rollback (sp): When we call rollback on save point instance, the transaction before the save point is saved but the transaction after the save point is not performed.

2.10.2 Batch Updations

- A batch update is a batch of updates grouped together, and sent to the database in one "batch", rather than sending the updates one by one. Sending a batch of updates to the database in one go, is faster than sending them one by one, waiting for each one to finish.
- There is less network traffic involved in sending one batch of updates (only 1 round trip), and the database might be able to execute some of the updates in parallel. The speed up compared to executing the updates one by one, can be quite big.
- We can batch both SQL inserts, updates and deletes. It does not make sense to batch select statements.
- There are **two ways to execute batch updates** :
  1. Using a Statement
  2. Using a PreparedStatement

1. Using StatementBatch Updates

- We can use a Statement object to execute batch updates. You do so using the addBatch() and executeBatch() methods. Here is an example:

```
Statement st = null;
try{
        st = connection.createStatement();
        st.addBatch("update customer set cname='Heet' where id=3454");
        st.addBatch("update customer set cname='priya' where id=47856");
        st.addBatch("update customer set cname='umang' where id=78879");
        int[] recordsAffected = st.executeBatch();

    }
finally
```



```
{
    if(st != null) st.close();
}
```

- First we add the SQL statements to be executed in the batch, using the addBatch() method. Then we execute the SQL statements using the executeBatch(). The int[] array returned by the executeBatch() method is an array of int telling how many records were affected by each executed SQL statement in the batch.

2. Using PreparedStatement Batch Updates

- We use a PreparedStatement object to execute batch updates. It enables us to reuse the same SQL statement, insert new parameters, for each update to execute. Let's see one example.

☞ **Example**

```
String sql = "update customer set cname=? , city=? where id=?";
PreparedStatement pst = null;
try{
        pst =connection.prepareStatement(sql);
        pst.setString(1, "Heet");
        pst.setString(2, "New Jersey");
        pst.setLong(3,3454 );
        pst.addBatch();
        pst.setString(1, "priya");
        pst.setString(2, "Canada");
        pst.setLong  (3, 47856);
        pst.addBatch();
        int[] affectedRecords = pst.executeBatch();
    }
    finally
    {
      if(pst != null) { pst.close();  }
    }
```

- First a PreparedStatement is created from an SQL statement with question marks in, to show where the parameter values are to be inserted into the SQL.
- Second, each set of parameter values are inserted into the pst, and the addBatch()method is called. This adds the parameter values to the batch internally. You can now add another set of values, to be inserted into the SQL statement. Each set of parameters are inserted into the SQL and executed separately, once the full batch is sent to the database.

- Third, the executeBatch() method is called, which executes all the batch updates. The SQL statement plus the parameter sets are sent to the database in one go. The int[] array returned by the executeBatch() method is an array of int telling how many records were affected by each executed SQL statement in the batch.

Syllabus Topic : BLOB and CLOB

2.11 JDBC BLOB, CLOB

Q. Write a short note on Blob object.

Q. Write a short note on Clob object.

- Blob, Clob Java objects can be manipulated without having to bring all of their data from the database server to your client computer.
- We can represent an instance of these types with a **locator** (logical pointer) to the object in the database that the instance represents because a BLOB, CLOB SQL object may be very large, the use of **locators** can make performance faster.
- To bring the data of a BLOB, CLOB SQL value to the client computer, use methods in the Blob, Clob Java interfaces.

☞ **Java.sql.BLOB interface methods**

1. **InputStreamgetBinaryStream() :** Retrieve the entire BLOB as a stream.
2. **byte[] getBytes**(long pos, int length) **:** Return a copy of the contents of the BLOB at the requested position.
3. **long position**(byte[] pattern, long start) **:** Determine the byte position at which the given byte pattern.
4. **long length() :** The length of the Binary Large OBject in bytes.
5. **long position**(Blob pattern, long start) **:** Determine the byte position at which the given pattern.

☞ **Java.sql.CLOB interface methods**

1. **InputStream getAsciiStream() :** It retrieves the CLOB value designated by this Clob object as an ascii stream.
2. **Reader getCharacterStream() :** It retrieves the CLOB value designated by this Clob object as java.io.Reader object (or as a stream of characters).

3. **Reader getCharacterStream(long pos, long length) :** It returns a Reader object that contains a partial Clob value, starting with the character specified by pos, which is length characters in length.
4. **String getSubString(long pos, int length) :** It retrieves a copy of the specified substring in the CLOB value designated by this Clob object.
5. **long length() :** It retrieves the number of characters in the CLOB value designated by this Clob object.
6. **Long position**(Clob searchstr, long start) **:** It retrieves the character position at which the specified Clob object searchstr appears.
7. **Long position**(String searchstr, long start) **:** It retrieves the character position at which the specified substring searchstr appears in the SQL CLOB value represented by this Clob object.

2.12 Programming Exercise

1. Create a table Employee having fields empid, empname and salary. Insert the records into the table dynamically. Write a java program that uses JDBC API to achieve the above.
2. Write a JDBC program that inserts dynamic values into a student table. The structure of student table is (rollno,student_name,marks)
3. Write a JDBC program to insert five records in customer table with field's custerno, firstname, lastname, address, mobilenumber and email.
4. Write a JDBC program that will display eighth record from employee table (empid, name, deptid, address).

❑❑❑