UNIT-1
JDBC

1. Introduction to JDBC:

JDBC stands for Java Database Connectivity, which is a standard Java API for database.

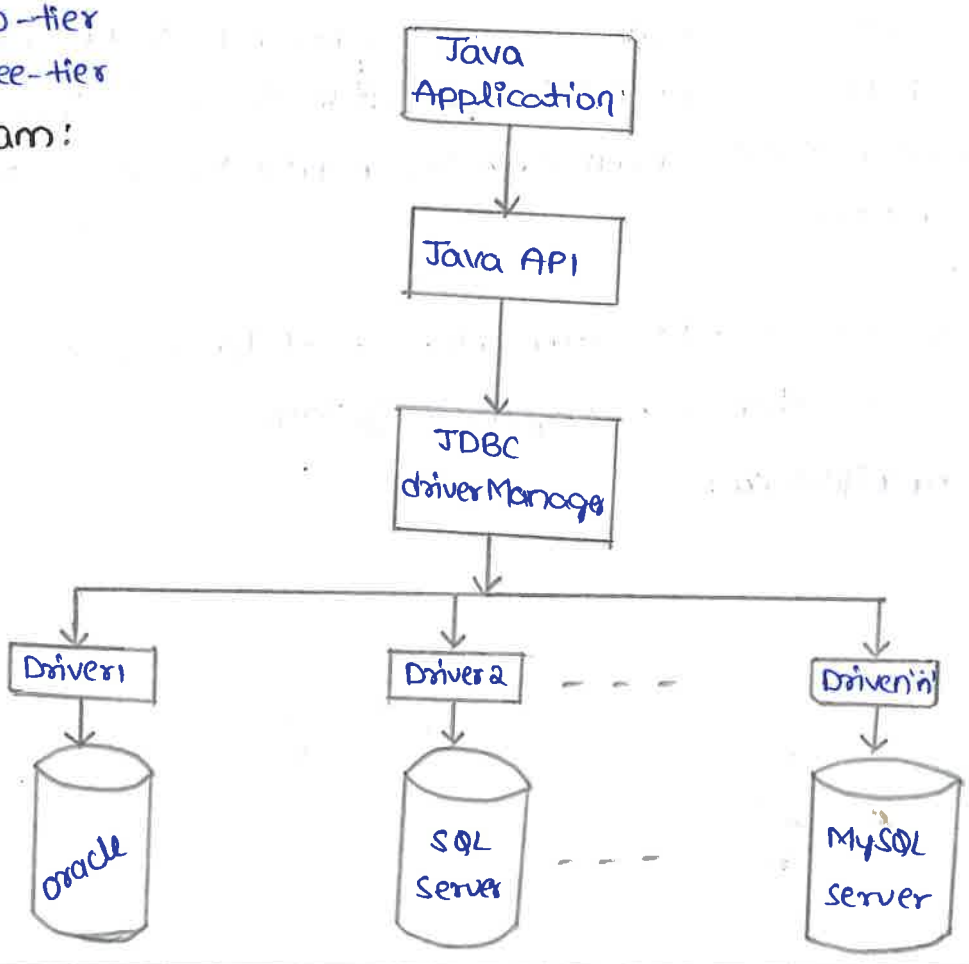
* JDBC is an API that helps applications to communicate with database's.
* JDBC allows Java programs to connect to a database, run queries, retrieve and manipulate data.
* Java applications can easily work with different relational databases like SQL, MySQL, oracle etc---

JDBC Architecture:

The architecture of JDBC in Java defines how Java applications interact with databases. It primarily involves several key components and supports two main processing models:

1. two-tier
2. three-tier

Diagram:



Components of JDBC Architecture:

Application:

It can be a Java application or servlet that communicates with a datasource.

JDBC API:

This is the core set of Java interfaces and classes (e.g., Connection, Statement, ResultSet, DriverManager) that provide the standard way for Java applications to connect to databases, execute SQL queries and process results.

JDBC DriverManager:

This class is responsible for loading and managing JDBC drivers. It acts as a central point for finding the appropriate driver to establish a connection with a specific database.

JDBC Driver:

These are software components that implement the JDBC API for a particular database system. They translate the generic JDBC calls into database-specific commands and handle the actual communication with the database.

Database:

This is the actual data source where the data resides and is managed by a database management system.

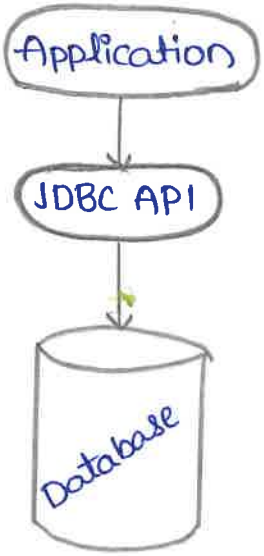
Types of Architecture:

1. 2-tier
2. 3-tier

Two-tier:

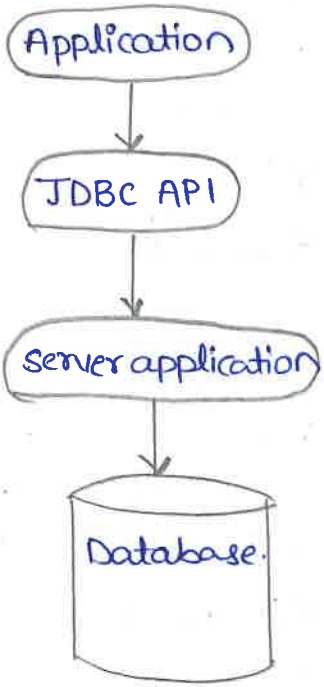
An application communicates directly with the database using a JDBC driver.

* It sends queries to the databases and then the result is send back to the application.



* This is suitable for simpler applications or where the database is located on the same machine or a directly accessible remote server.

Three-tier



* In this, user queries are sent to a middle-tier services, which interacts with the database.
* The database results are processed by the middleware.

JDBC Steps:

1. import packages
2. load and register the drivers
3. Establish connection
4. create a statements
5. update
6. Result
7. close the connection.

Program:

```
package tec;
import java.sql.*;
Public class IT
{
 public static void main(String[] args) throws Exception
 {
  Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con = DriverManager.getConnection("jdbc:oracle:thin:
        @localhost:1521:XE","system","tec");
Statement stmt = con.createStatement();
stmt.executeUpdate("create table student(Name varchar(20),
                    age int)");
stmt.executeUpdate("insert into student values('Rama',03)");
stmt.executeUpdate("insert into student values('sita', 02)");
stmt.executeUpdate("insert into student values('Hanuma',04)");
stmt.executeUpdate("insert into student values('Lakshmana',05)");
stmt.executeUpdate("update Lakshmana set Name='Beema'
                    where age=5");
```

```
stmt.executeUpdate("delete from Studat where age=4");
ResultSet rs = stmt.executeQuery("select * from Studat");
while(rs.next())
{
 System.out.println(rs.getString(1)+" "+rs.getInt(2));
 }
 rs.close();
 stmt.close();
 con.close();
 }
}
```

Output:

Name	Age
Rama	3
Sita	2
Beema	5

2. JDBC Drivers:

JDBC drivers are client-side adapters that translate requests from Java program into a protocol understand by the DBMS

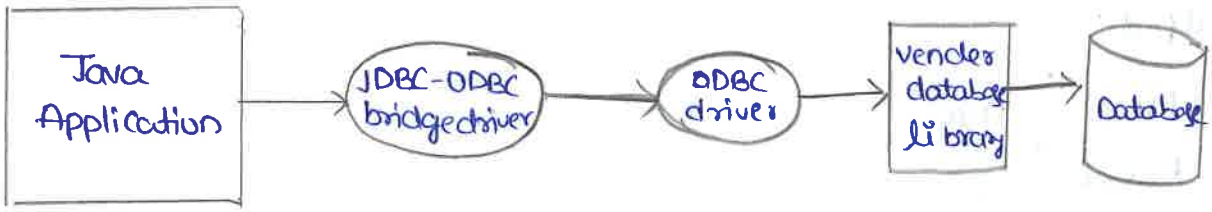
Types of drivers:

1. JDBC-ODBC bridge drivers (or) Type-I
2. Type-II (or) Native-API drivers
3. Type-III (or) Network protocol drivers
4. Type-IV (or) Thin drivers.

1. Type-I (or) JDBC-ODBC bridge drivers:

It uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls.

* It is also called Universal drivers because it can be used to connect to any of the database
* This type is often considered outdated, requires an ODBC driver on the client, and is not recommended for most modern application.



Advantage:

* This driver software is built-in with JDK so no need to install seperately
* It is a database independent driver.

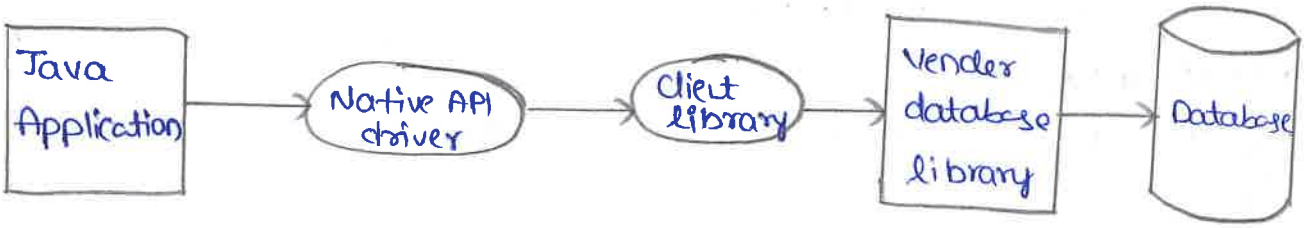
Disadvantage:

* The ODBC bridge driver is needed to be installed in individual client machines.
* Not using
* No security.

2. Type-II (or) Native-API Driver:

The Native API driver uses the client-side libraries of the database. This driver converts JDBC method calls into native calls of the database API. In order to interact with different databases.

* This driver needs their local API, that's why data transfer is much more secure as compared to type-1 driver.
* This driver is not fully written in Java that is why it's also called Partially Java driver.



Advantage:

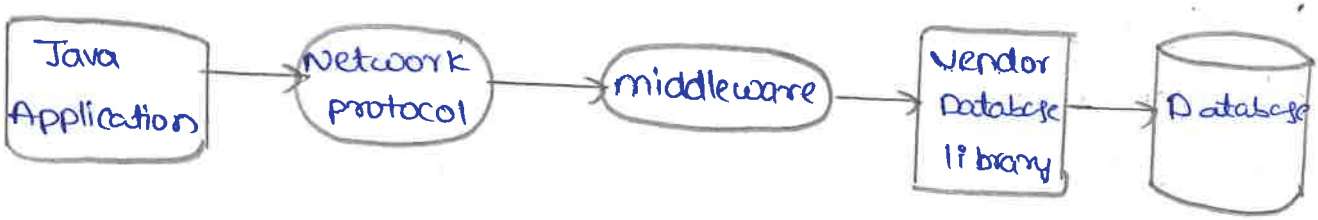
* Native-API driver gives better performance than JDBC-ODBC bridge drivers
* More secure

Disadvantage:

* Need to be installed seperately in individual client side libraries
* It is a database dependent driver

3. Type-III (or) Network Protocol Driver:

* The network protocol driver uses middleware that converts JDBC calls directly or indirectly into the vendor-specific database protocol.
* In this all the databases connectivity drivers are present in single server. hence, no need of individual client-side installation.



Advantage:

* Connect multiple databases at a time.
* No client side library is required because of application server.
* Easy to switch databases

Disadvantage:

* Network support is required on client machine
* More cost
* Complex Architecture

4. Type-IV (or) Thin driver:

Type-IV driver is also called native protocol driver.

* This driver interact directly with database.
* It doesnot require any native database library, that is why it is also known as Thin drivers.

Advantage:

* Doesnot require any native library and middleware server.
* Faster
* high performance
* efficiency.

Disadvantage:

* At a time it connect to only one database
* If the database changes, a new driver may be needed.

program:

```
package com.jdbc;
import java.sql.*;
public class Jdbc.
{
 public static void main(String[] args) throws Exception
 {
  Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con = DriverManager.getConnection("jdbc:oracle:thin:
            @localhost:1521:XE","system","tec");
Statement s = con.createStatement();
s.executeUpdate("create table Student(Name varchar(20),age int)");
s.executeUpdate("insert into Student values('Rama',03)");
ResultSet rs = s.executeQuery("select * from Student");
 while(rs.next())
 {
  System.out.println(rs.getString(1)+" "+rs.getInt(2));
 }
 rs.close();
  s.close();
 con.close();
  }
}
```

Output:

Rama 3

3. Statements in JDBC:

In JDBC, a statement object is an interface used to create and execute SQL queries and updates against a relational database from a Java application. It acts as a bridge between Java code and the database.

*There are three main types of Statement objects in JDBC.

1. Statement
2. prepared Statement
3. Callable Statement.

Statement:

A statement object is used for general purpose, used to access the data to database.

* It is useful for executing static SQL statements at runtime.
* This is the most basic type, used for executing static SQL statements.
* It is suitable for one-time queries or updates that do not involves user input, as it directly takes the SQL Statements as a string.

Syntax:

```
Statement stmt = connection.createStatement();
```

program:

```
package com.jdbc;
import java.sql.*;
public class Jdbc
{
 public static void main(String[] args) throws Exception
 {
  Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con = DriverManager.getConnection("jdbc:oracle:thin:
        @localhost:1521:XE","system","tec");
 Statement stmt = con.createStatement();
```

```
stmt.executeUpdate("create table Student(Name char(20), age int)");
stmt.executeUpdate("insert into Student values ('Rama', 03)");
stmt.executeUpdate("insert into Student values ('sita', 04)");
ResultSet rs = stmt.executeQuery("select * from Student");
while(rs.next())
{
 System.out.println(rs.getString(1)+" "+rs.getInt(2));
}
rs.close();
stmt.close();
con.close();
  }
}
```

Output:

Name	Age
Rama	3
Sita	3

Prepared Statement:

It represents a precompiled SQL statements that can be executed multiple times.

* It accepts parameterized SQL queries, with "?" as a placeholder for parameters.
* It offers better performance for repeated executions of the same query with different parameters.

Syntax:

```
PreparedStatement stmt = con.prepareStatement("insert into
                    emp(?, ?)");
stmt.setString(1, "John");
stmt.setInt(2, 30);
stmt.executeUpdate();
```

program:

```
Package com.jdbc;
import java.sql.*;
Public class Jdbc
{
public static void main(String args[]) throws Exception
{
  Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost
        :1521:XE", "system","tec");
Statement stmt = con.createStatement();
stmt.executeUpdate("create table A(rollno int primarykey,name char(20),
                                   address char(20))");
PreparedStatement ps = con.preparedStatement("insert into A values(?,?,?)");
 ps.setInt(1,001);
 ps.setString(2,"Rama");
 ps.setString(3,"CPT");
ResultSet rs = ps.executeQuery("select * from A");
while(rs.next())
{ System.out.println(rs.getString(1)+" "+rs.getInt(2));
}rs.close();
  ps.close();
  stmt.close();
  con.close();
}}
```

Output: 001 Rama CPT

Callable Statement: This type extends preparedStatement and is used for executing stored procedures in the database.

* It allows you to call stored procedures and retrieve their out put parameters or return values

Syntax:

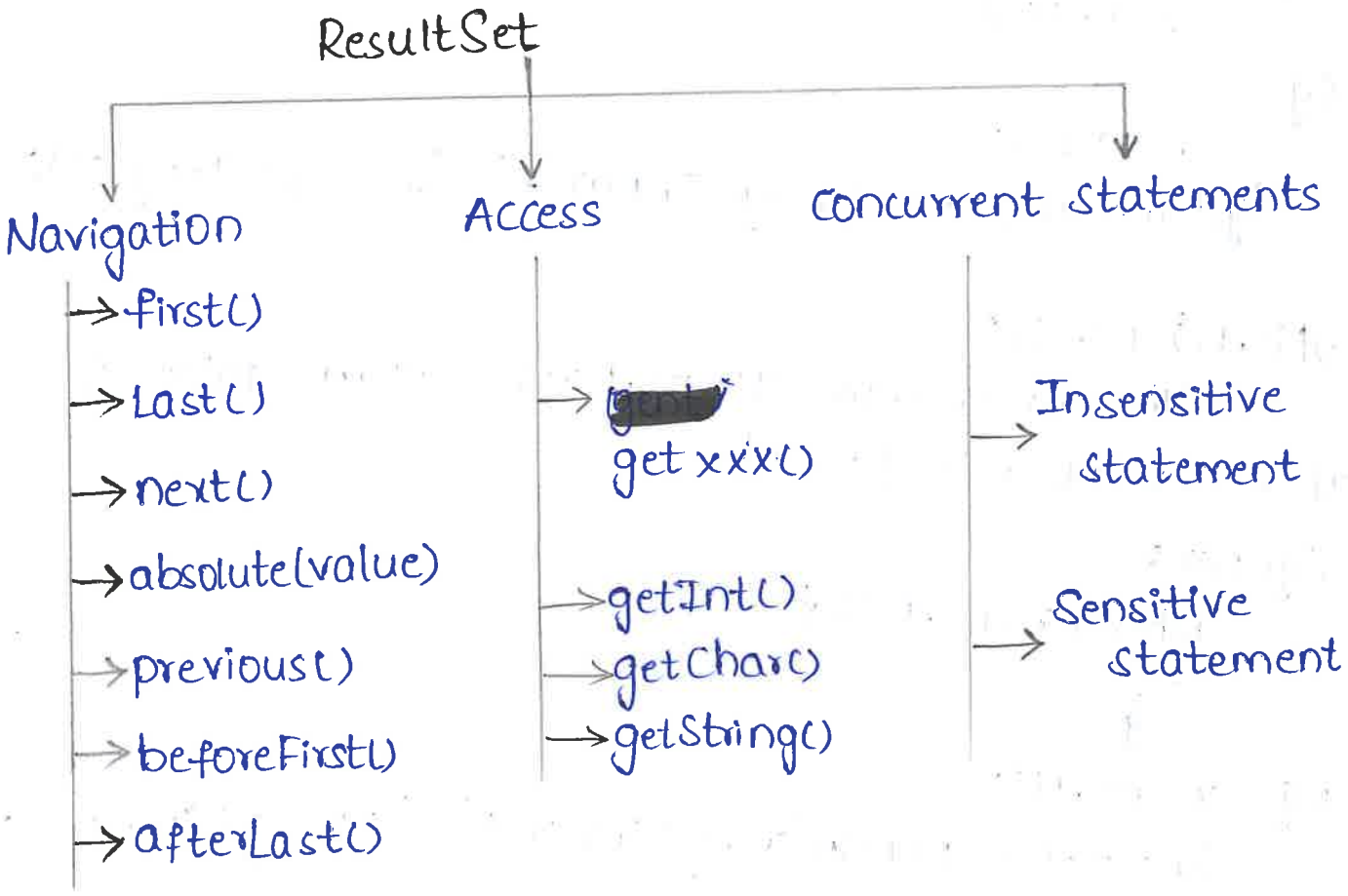
```
CallableStatement cstmt = con.prepareCall("{call procedurename
                                            (?,?)}");
cstmt.setInt(1,101);
cstmt.execute();
```

ResultSet operations:-

ResultSet operations are used to retrieve the data from the database query

⇒ These are classified into 3 types

1) Navigation
2) Access
3) Concurrent Statements



Navigation:-

It is a sets of rules for choosing the next page to displayed after a button or hyperlink is clicked.

⇒ In this some operations are there.

1) first():-

Used to move to first row (or) record in the ResultSet

Syntax:-

```
rs.first();
```

Eg:-

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

2) last():-

Used to move to Last row (or) record in the ResultSet

Syntax:-

```
rs.last();
```

Eg:-

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

3) Next() next():-

The cursor move to next row (or) record, return false if there are no rows present

Syntax:-

```
While (rs.next()) {

    }
```

Eg:-

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

4) absolute(value):-

The cursor moves to Specific (or) particular row (or) record

Syntax:-

```
rs.absolute(value);
```

Eg:- rs.absolute(2);
System.out.println(rs.getInt(1) + " " + rs.getString(2));

4) Previous() :-
The cursor moves to the previous row(or) record

Syntax:-
rs.previous();

Eg:- rs.previous();
System.out.println(rs.getInt(1) + " " + rs.getString(2));

5) beforeFirst():-
The cursor moves to the position before the first row(or) record

Syntax:-
rs.beforeFirst();

Eg:- rs.beforeFirst();
System.out.println(rs.getInt(1) + " " + rs.getString(2));

6) afterLast():-
The cursor moves to the position after the lastrow

syntax:-
rs.afterLast();

Eg:-
rs.afterLast();
System.out.println(rs.getInt(1) + " " + rs.getString(2));

Access:-

⇒ It also used to retrieve the data

⇒ In this data we use getxxx() method

⇒ If the data is in

Integer type ⟶ getInt()

character type ⟶ getChar()

float type ⟶ getFloat()

String type ⟶ getString()

⇒ Here "xxx" represent datatype

Syntax:-

getInt (columnindex value)

Concurrent Statements:-

⇒ These are also used to retrieve the data

⇒ There are two types

⇒ InSensitive statement

⇒ Sensitive statement

1) Insensitive statement:-

⇒ It is used for read only, no modifications, no updates are possible

⇒ In this we just read the data

Eg:- Google sites

Syntax:-

Statement stmt = connection. createStatement (ResultSet. TYPE_SCROLL_INSENSITIVE, ResultSet. CONCUR_READ_ONLY);

Sensitive:-

It is used to update the data, modifications are possible

Eg:- Uses programs

Syntax:-

Statement stmt = con.createstatement(ResulSet.TYPE_SCROLL-SENSITIVE, ResultSet.CONCOR_UPDATABLE);

Program:-

```
Creat table studentit1222 (rollno int, name char(30), age int);
insert into studentit1222 values(1,'rama',03);
insert into studentit1222 values(2,'sita',02);
insert into studentit1222 values(3,'Lakshmana',04);
insert into studentit1222 values(4,'hanuma',05);
Select * from studentit1222;

package operations;
import java.sql.*;
public class Rs {
 public static void main(String[] args)throws Exception
{
 Class.forName("Oracle.jdbc.driver.OracleDriver");
Connection con = DriverManager.getConnection ("jdbc:oracle:
thin:@localhost:1521:XE", "system", "tec");
Statement stmt = con.createStatement( Resultset.TYPE_SCROLL_
INSENSITIVE, ResultSet.CONCUR_READ_ONLY);
```

```
ResultSet rs = stmt.executeQuery("select * from studentit
1222");
rs.first();
System.out.println(rs.getInt(1)+" "+rs.getString(2) rs.getStri
                   +" "+rs.getInt(3));

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

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

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

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

rs.close();
stmt.close();
Con.close();
}}
```

output:-

Rollno	Name	Age
1	rama	3
2	Sita	2
3	Lakshmana	4
4	Hanuma	5

Rollno	Name	Age
1	rama	3
24	hanuma	5
2	Sita	2
3	Lakshmana	4
2	Sita	2

Batch Update:-

Group of commands is known as BatchUpdate

⇒ The commands are insert (or) update (or) delete

⇒ We use batch update concept because no.of insert commands are executed at a time.

Advantage:-

⇒ Performance will be increases

⇒ Network overload will be decreases

⇒ In this we can add the stmt to batch by using

```
Stmt.addBatch();
```

⇒ For executing the batch

```
Stmt.executeBatch();
```

In BatchUpdate, first we add the data or gather all the data together and then we execute the stmt

⇒ Through batch update,
many no.of stmts are executed at a time

⇒ It saves the time network.

Program:-

```
Create table studentit1222 (rollno int, name char(30),
                            age int);
insert into studentit1222 values (1, 'rama', 03);
insert into studentit1222 values (2, 'sita', 02);
insert into studentit1222 values (3, 'Lakshmana', 04);
insert into studentit1222 values (4, 'hanuma', 05);
Select * from studentit1222;
```

```
package operations;
import java.sql.*;
public class Rs {
 public static void main(String[] args) throws Exception
{
Class.forName("oracle.jdbc.driver.OracleDriver");
Con.setAutoCommit(false);
Statement stmt = con.createstatement (ResultSet.TYPE_
SCROLL_SENSITIVE . ResultSet.CONCUR_UPDATABLE);
Resultset rs= stmt.executeQuery ("select age from studentit
1222);
rs.first();
int newage = rs.getInt("age")+1;
rs.updateRow();
System.out.println(newage);
rs.close();
stmt.close();
con.commit();
Con.close();
}}
```

Output:-

Rollno	Name	Age
1	rama	3
2	sita	2
3	Lakshmana	4
4	Hanuma	5

RollNo	Name	Age	RollNo	Name	Age
1	rama	3	1	rama	4
2	Sita	2	2	Sita	2
3	lakshmana	4	3	Lakshmana	4
4	Hanuma	5	4	Hanuma	5

Managing DataBase Transactions :-

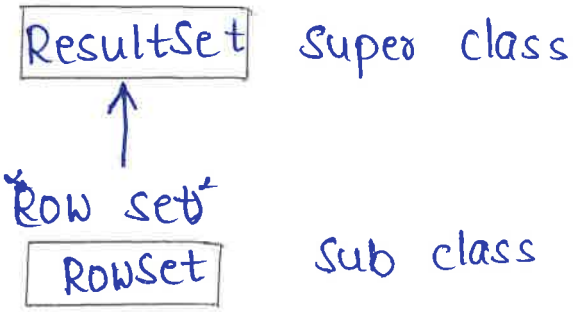
⇒ Multiple SQL queries are executed at a time

⇒ Transaction contains multiple SQL Queries

⇒ In multiple SQL queries, if one query is having exception the remaining queries are also not executed so to handle that exception we use try & catch blocks

RowSet :-

It is derived from resultset



⇒ RowSet is more powerful, because it is having Result row set properties

In this we use CRUD

C- Create

R- Read

U - Update

D - Delete