**1.Definition:-** Process of combining all related operations into single unit and executing on the rule “either all or none”, is called transaction management.

Ex1:- Transaction Name: Fund Transfer.

Operation1: Debit fund from sender account.

Operation2:credit fund into receiver account.

Ex2:- Transactiion Name: Movie Ticket Reservation.

Operation1:Verify the Status.

Operation2:Reserve the tickets.

Operation3:Payment.

Operation4:issue Ticket.

All operations in Transaction should be performed as single unit only. If some operations success and some operations fails then there may be data inconsistency problem. Therefore every transaction must work on rule “either all or none” to avoid data inconsistency problem.

**2.Transaction Types:-** There are two types of Transactions.

9.2.1) Local Transaction.

9.2.2) Global Transaction.

2.1) Local Transaction:- if all operations in transaction are executed over same database, then that transaction is called Local Transaction.

Ex:- Fund transfer from one account to another account where both accounts in same bank.

Note:- JDBC supports Local Transaction.

2.2) Global Transaction:- If transaction is executed over different databases , then

That transaction is called Global Transaction.

Ex:-Fund transfer from one account to another account where both accounts in different banks.

3. JDBC manages transaction with following four methods.

1.setAutoCommit(false/true).

2.commit();

3.rollback();

All above methods belongs to **Connection** Interface.

Example:-1 App demonstrates the Transaction Management.

import java.sql.ResultSet;

import oracle.jdbc.pool.OracleConnectionPoolDataSource;

import java.util.Scanner;

class Type4

{

public static void main(String arg[])

{

try

{

OracleConnectionPoolDataSource ds=new OracleConnectionPoolDataSource();

ds.setURL("jdbc:oracle:thin:@localhost:1521:ORCL");

ds.setUser("scott");

ds.setPassword("sukumar");

Connection con=ds.getConnection();

Statement p=con. createStatement();

ResultSet r=p.executeQuery("select \* from sukumar");

System.out.println("------Data Before Transaction-----");

while(r.next())

System.out.println(r.getInt(1)+" "+r.getInt(2));

System.out.println("---------------Transaction Begins--------");

con.setAutoCommit(false);

p.executeUpdate("update sukumar set balance=balance-1000 where ano=1");

p.executeUpdate("update sukumar set balance=balance+1000 where ano=2");

System.out.println("Do you commit Transaction:");

Scanner sv=new Scanner(System.in);

String ch=sv.next();

if(ch.equalsIgnoreCase("yes"))

con.commit();

else

con.rollback();

r=p.executeQuery("select \* from sukumar");

System.out.println("------Data After Transaction-----");

while(r.next())

System.out.println(r.getInt(1)+" "+r.getInt(2));

}

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

}

}

Output:

--------

D:\>java Type4

------Data Before Transaction-----

1 7000

2 5000

---------------Transaction Begins--------

Do you commit Transaction:

yes

------Data After Transaction-----

1 6000

2 6000

Example:2 App demonstrates batch processing + transaction processing.

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

public class Sample {

public static String qur="select eno,ename from emp";

public static void main(String[] args) throws InterruptedException {

try(Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","sukumar","sukumar");

Statement s=con.createStatement();

){

if(s!=null && con!=null) {

con.setAutoCommit(false);

s.addBatch("update emp set ename='sula' where eno=6");

s.addBatch("insert into emp values(8,'sulamaha')");

int []result=s.executeBatch();

int i=0,resultflag=0;

while(i<result.length) {

if(result[i]==0)

resultflag=1;

i=i+1;

}

if (resultflag==0) {

con.commit();

System.out.println("Transaction Completed");

}

else {

con.rollback();

System.out.println("Transaction Failed");

}

}

}

catch(SQLException e) {

System.out.println(e.getMessage());

}

}

}

Output:

---------

Transaction completed.

4. SavePoint:- It is an interface in java.sql package. Driver software vendor is responsible to provide implementation.Savepoint concept is applicable only in transactions.

The following methods are used for managing savepoint.

1.setSavepoint()

2.releaseSavepoint()

3.rollback(Savepoint var);

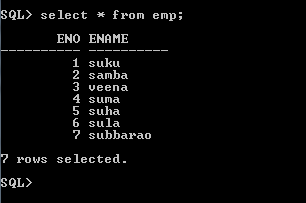
All above methods belongs to Connection Interface.

Only few drivers supports the save point.

Type1- does not support save point.

Type4- driver of Oracle supports the only for setSavepoint() and rollback() methods but not for releaseSavepoint() method.

Example:-



package com.raos.jdbc;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Savepoint;

import java.sql.Statement;

public class Sample {

public static String qur="select eno,ename from emp";

public static void main(String[] args) throws InterruptedException {

try(Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","sukumar","sukumar");

Statement s=con.createStatement();

){

if(s!=null && con!=null) {

con.setAutoCommit(false);

s.executeUpdate("update emp set ename='sulakshmi' where eno=6");

Savepoint sv=con.setSavepoint("sv");

s.executeUpdate("insert into emp values(8,'sulamaha')");

con.rollback(sv);

con.releaseSavepoint(sv);

System.out.println("Operations are rollback from savepoint");

}

}

catch(SQLException e) {

System.out.println(e.getMessage());

}

}

}

Output:

---------

Operations are rollback from savepoint

