Transaction Management in JDBC
==========
=> Process of combining all related operations into a single unit and executing on the rule
     "either all or none", is called transaction management.
=> Hence transaction is a single unit of work and it will work on the rule "either all or none".

Case-1: Funds Transfer
1. debit funds from sender's account
2. credit funds into receiver's account
All operations should be performed as a single unit only. If debit from sender's account completed
and credit into receiver's account fails then there may be a chance of data inconsistency problems.

Case-2: Movie Ticket Reservation
1. Verify the status
2. Reserve the tickets
3. Payment
4. issue tickets.
All operations should be performed as a single unit only. If some operations success and some operations fails then
there may be data inconsistency problems.

Transaction Properties:
Every Transaction should follow the following four ACID properties.
1. A → Atomiticity
   Either all operations should be done or None.
2. C → Consistency(Reliabile Data)
   It ensures bringing database from one consistent state to another consistent state.
3. I → isolation (Seperatation)
   Ensures that transaction is isolated from other transactions
4. D → Durability
   It means once transaction committed, then the results are permanent even in the case of system restarts, errors etc.

Types of Transactions
==========
There are two types of Transactions
1. Local Transactions
2. Global Transactions

1. Local Transactions:
     All operations in a transaction are executed over same database.
     Eg: Funds transfer from one accoun to another account where both accounts in the same bank.
2. Global Transactions:
    All operations is a transaction are expected over different databases.
    Eg: Funds Transfer from one account to another account and accounts are related to
        different banks.

Note:
JDBC can provide support only for local transactions.
If we want global transactions then we have to go for EJB(Enterprise Java Bean) or Spring framework.

Process of Transaction Management in JDBC:
1. Disable auto commit mode of JDBC
   By default auto commit mode is enabled. i.e after executing every sql query, the changes will be committed automatically
   in the database.
   We can disable auto commit mode as follows
            con.setAutoCommit(false);

2. If all operations completed then we can commit the transaction by using the following method.
            con.commit();

3. If any sql query fails then we have to rollback operations which are already completed by using rollback() method.
            con.rollback();

Program to demonstrate Transaction app
Savepoint(I)
============
=> Savepoint is an interface present in java.sql package.
=> Introduced in JDBC 3.0 Version.
=> Driver Software Vendor is responsible to provide implementation.
=> Savepoint concept is applicable only in Transactions.
=> Within a transaction if we want to rollback a particular group of operations based on some
      condition then we should go for Savepoint.
=> We can set Savepoint by using setSavepoint() method of Connection interface.
            Savepoint sp = con.setSavepoint();
=> To perform rollback operation for a particular group of operations wrt Savepoint, we can use rollback() method as follows.
            con.rollback(sp);
=> We can release or delete Savepoint by using release Savepoint() method of Connection interface.
                  con.releaseSavepoint(sp);


```
con.setAutoCommit(false)
operation-1
operation-2
operation-3
SavePoint sp =new SavePoint();
operation-4
operation-5
      if(balance<=1000)
            con.rollback(sp);
      else
            con.releaseSavePoint();
operation-6
con.commit();
```

At line-1 if balance <10000 then operations 4 and 5 will be Rollback, otherwise all operations will be performed normally.

Note:
Some drivers won't provide support for Savepoint. Type-1 Driver won't provide support, but Type#4 Driver can provide support.

Type-4 Driver of Oracle provide support only for setSavepoint() and rollback() methods but not for
releaseSavepoint() method.

Transaction Concurrency Problems:
Whenever multiple transactions are executing concurrently then there may be a chance of
transaction concurrency problems.

The following are the most commonly occurred concurrency problems.
1. Dirty Read Problem
2. Non Repeatable Read Problem
3. Phantom Read Problem

1. Dirty Read Problem:
Also known as uncommitted dependency problem.
Before committing the transaction, if its intermediate results used by any other transaction then
there may be a chance of Data inconsistency problems. This is called Dirty Read Problem.

nitin:50000
T1:update accounts set balance=balance+50000 where name='nitin'
T2:select balance from accounts where name='nitin'
T1: con.rollback();

At the end, T1 point of view, nitin has 50000 balance and T2 point of view nitin has 1Lakh. There
may be a chance of data inconsistency problem. This is called Dirty Read Problem.

2. Non-Repeatable Read Problem:
For the same Read Operation, in the same transaction if we get different results at different times,then such type of problem is called Non-Repeatable Read Problem.
Eg:
T1: select * from employees;
T2: update employees set esal=10000 where ename='nitin';
T1: select * from employees;
In the above example Transaction-1 got different results at different times for the same query.

3. Phantom Read Problem:
A phantom read occurs when one transaction reads all the rows that satisfy a where condition and
second transaction insert a new row that satisfy same where condition. If the first transaction
reads for the same condition in the result an additional row will come.
This row is called phantom row and this problem is called phantom read problem.

T1: select * from employees where esal >5000;
T2: insert into employees values(300,'ravi',8000,'hyd');
T1: select * from employees where esal >5000;
In the above code whenever transaction-1 performing read operation second time, a new row will
come in the result.

To overcome these problems we should go for Transaction isolation levels.

Connection interface defines the following 4 transaction isolation levels.
1. TRANSACTION_READ_UNCOMMITTED → 1
2. TRANSACTION_READ_COMMITTED → 2
3. TRANSACTION_REPEATABLE_READ → 4
4. TRANSACTION_SERIALIZABLE → 8

1. TRANSACTION_READ_UNCOMMITTED:
   It is the lowest level of isolation.
   Before committing the transaction its intermediate results can be used by other transactions.
   Internally it won't use any locks.
   It does not prevent Dirty Read Problem, Non-Repeatable Read Problem and Phantom Read Problem.
   We can use this isolation level just to indicate database supports transactions.
   This isolation level is not recommended to use.

2. TRANSACTION_READ_COMMITTED:
   This isolation level ensures that only committed data can be read by other transactions.
   It prevents Dirty Read Problem. But there may be a chance of Non Repeatable Read Problem and
   Phantom Read Problem.

3. TRANSACTION_REPEATABLE_READ:
    This is the default value for most of the databases. Internally the result of SQL Query will be locked for only one transaction. If we perform multiple read operations, then there is a guarantee that for same result.
     It prevents Dirty Read Problem and Non Repeatable Read Problems. But still there may be a
     chance of Phantom Read Problem.

4. TRANSACTION_SERIALIZABLE:
    It is the highest level of isolation.
    The total table will be locked for one transaction at a time.
    It prevents Dirty Read, Non-Repeatable Read and Phantom Read Problems.
    Not Recommended to use because it may creates performance problems.

Note:
Connection interface defines the following method to know isolation level.
      getTransactionIsolation()
Connection interface defines the following method to set our own isolation level.
      setTransactionIsolation(int level)

Eg:
```
System.out.println(con.getTransactionIsolation());
con.setTransactionIsolation(8);
System.out.println(con.getTransactionIsolation());
```

Note:
For Oracle database, the default isolation level is: 2(TRANSACTION_READ_COMMITED).
      Oracle database provides support only for isolation levels 2 and 8.
For MySql database, the default isolation level is: 4(TRANSACTION_REPEATABLE_READ).
      MySql database can provide support for all isolation levels (1, 2, 4 and 8).

Note:
ResultSet(holds the data which is used for reading purpose)
      |=> Using resultset we have just performed read operation(best suited)
      |=> Is it possible to perform update,inserte and delete operation(possible

but not recomended)

RowSet(ALL DB vendors jar support for RowSet is not available)
==========
=> It is alternative to ResultSet.
=> We can use RowSet to handle a group of records in more effective way than ResultSet.
=> RowSet interface present in javax.sql package
=> RowSet is child interface of ResultSet.
=> RowSet implementations will be provided by Java vendor and database vendor.
=> By default RowSet is scrollable and updatable.
=> By default RowSet is serializable and hence we can send RowSet object across the network. But
ResultSet object is not serializable.
=> ResultSet is connected i.e to use ResultSet compulsary database Connection must be required.
=> RowSet is disconnected. ie to use RowSet database connection is not required.

Types of RowSets
==========
There are two types of RowSets
1. Connected RowSets
2. Disconnected RowSets

Connected RowSets
==========
Connected RowSets are just like ResultSets.
To access RowSet data compulsary connection should be available to database.
We cannot serialize Connected RowSets.
Eg: JdbcRowSet

Disconnected RowSets:
Without having Connection to the database we can access RowSet data.
We can serialize Disconnected RowSets.
Eg:
CachedRowSet
WebRowSet
a.FilteredRowSet
b.JoinRowSet
How to create RowSet objects?
We can create different types of RowSet objects as follows
```
RowSetFactory rsf = RowSetProvider.newFactory();
JdbcRowSet jrs = rsf.createJdbcRowSet();
CachedRowSet crs = rsf.createCachedRowSet();
WebRowSet wrs = rsf.createWebRowSet();
JoinRowSet jnrs = rsf.createJoinRowSet();
FilteredRowSet frs = rsf.createFilteredRowSet();
```

JdbcRowSet
==========
=> It is exactly same as ResultSet except that it is scrollable and updatable.
=> JdbcRowSet is connected and hence to access JdbcRowSet compulsary Connection must be
required.
=> JdbcRowSet is non serializable and hence we cannot send RowSet object across the network.

Note:

```
      jdbcRowSet.setUrl("jdbc:mysql:///abc");
      jdbcRowSet.setUser("root");
      jdbcRowSet.setPassword("root123");
      jdbcRowSet.setCommand("select eid,ename,esal,eaddress from employee");
      jdbcRowSet.execute();
```

Application to demonstrate
1. Retreive records from jdbcRowSet
2. Insert records into jdbcRowSet
3. Update record into jdbcRowSet
4. delete record into jdbcRowSet

CachedRowSet:
=> It is the child interface of RowSet.
=> It is bydefault scrollable and updatable.
=> It is disconnected RowSet. ie 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 DBConnection.
=> If we perform any update operations(like insert,delete and update) to the CachedRowSet,to
reflect those changes compulsary Connection should be established.
=> Once Connection established then only those changes will be reflected in Database.

Application to demonstrate
1. Retreive records from CachedRowSet
2. Insert records from CachedRowSet
3. Update record from CachedRowSet
4. delete record from CachedRowSet

Retreive a record
==========
1. Use Connection Object and get Statement,resultSet object
2. Get CachedRowSet Object and populate(resultSet) into CachedRowSet
3. use CachedRowSet to retreive the records.

Update record from CachedRowSet
==========
Make sure get the Connection Object with autocommit as false.
1. crs.setTableName(tableName);
2. crs.populate(resultSet)
3. crs.absolute(rowNo)
4. crs.updateString(2,ename); crs.updateFloat(3,esal); crs.updateString(4,eaddr);
5. crs.updateRow()
6. crs.acceptChanges(connection)

delete record from CachedRowSet
==========
Make sure get the Connection Object with autocommit as false.
1. crs.setTableName(tableName);
2. crs.populate(resultSet)
3. crs.last();

```
   4. crs.deleteRow();
   5. crs.acceptChanges(connection)

insert record into CachedRowSet
===============================
Make sure get the Connection Object with autocommit as false.
   1. crs.setTableName(tableName);
   2. crs.populate(resultSet)
   3. crs.moveToInsertRow();
   4. crs.updateNull(eid);//Autogenerated value
   5. crs.updateString(2,ename);crs.updateFloat(3,esal); crs.updateString(4,eaddr);
   6. crs.insertRow();
   7. crs.moveToCurrentRow();
   8. crs.acceptChanges(connection)

WebRowSet(I):
=> It is the child interface of CachedRowSet.
=> It is bydefault scrollable and updatable.
=> It is disconnected and serializable
=> WebRowSet can publish data to xml files,which are very helpful for enterprise
applications.
      FileWriter fw=new FileWriter("emp.xml");
      rs.writeXml(fw);

=> We can read XML data into RowSet as follows
      FileReader fr=new FileReader("emp.xml");
      rs.readXml(fr);

selecting the records
=====================
 1. rs.setCommand("select eid,ename,esal,eaddr from emp");
 2. rs.execute();
 3. FileWriter fw=new FileWriter("emp.xml");
 4.  rs.writeXml(fw);
 5.  rs.acceptChanges()

inserting the records
=====================
 1. rs.setCommand("select eid,ename,esal,eaddr from emp");
 2. rs.execute();
 3. FileReader fr=new FileReader("input.xml");
 4.  rs.readXml(fr);
 5.  rs.acceptChanges()

input.xml
=========
<data>
   <insertRow>
      <columnValue>11</columnValue>
      <columnValue>dupples</columnValue>
      <columnValue>RCB</columnValue>
      <columnValue>45</columnValue>
   </insertRow>
</data>

deleting the records
====================
 1. rs.setCommand("select eid,ename,esal,eaddr from emp");
 2. rs.execute();
```

```
 3. FileReader fr=new FileReader("input.xml");
 4.  rs.readXml(fr);
 5.  rs.acceptChanges()

input.xml
=========
<data>
   <deleteRow>
      <columnValue>11</columnValue>
      <columnValue>dupples</columnValue>
      <columnValue>RCB</columnValue>
      <columnValue>45</columnValue>
   </deletRow>
</data>

JoinRowSet:
==========
=> It is the child interface of WebRowSet.
=> It is by default scrollable and updatable
=> It is disconnected and serializable
=> If we want to join rows from different rowsets into a single rowset based on
matched
   column(common column) then we should go for JoinRowSet.
=> We can add RowSets to the JoinRowSet by using addRowSet() method.
    addRowSet(RowSet rs,int commonColumnIndex);

eg#1.
  CachedRowSet crs1=rsf.createCachedRowSet();
  crs1.setCommand("select sid,sname,saddr,cid from student");
  crs1.exeucte(con);

  CachedRowSet crs2=rsf.createCachedRowSet();
  crs2.setCommand("select cid,cname,cost from course");
  crs2.execute(con);

  JoinRowSet jrs=rsf.joinRowSet();
  rs.addRowSet(crs1,4);
  rs.addRowSet(crs2,1);

  //process the resultSet


FilteredRowSet(I):
=================
=> It is the child interface of WebRowSet.
=> If we want to filter rows based on some condition then we should go for
FilteredRowSet.
      public interface FilteredRowSet{
            public boolean evaluate(RowSet rs);//for filtering logic
            public boolean evaluate(Object obj,int colIndex);//for insertion of
record
            public boolean evaluate(Object obj,String colName);//for insertion of
record
      }

Note:
public boolean evaluate(RowSet rs){
      try {
            String colValue = rs.getString(colName);
```

```
            if (colValue.startsWith(condValue)) {
                  return true;
            } else {
                  return false;
            }
            } catch (SQLException e) {
                  e.printStackTrace();
            }

      }
}
```

Behind the scenes
==========
for every rs.next(), the entire record will be pulled and it will be given to RowSet(rs)
so from RowSet(rs) we need to get the ColValue based on ColName.
check the colValue with our condValue,if it matches return true, if it is true then that particular row will be available in rowSet.if not that rowSet will not be availabe for rendering.