Java Training Center

(No.1 in Training & Placement)

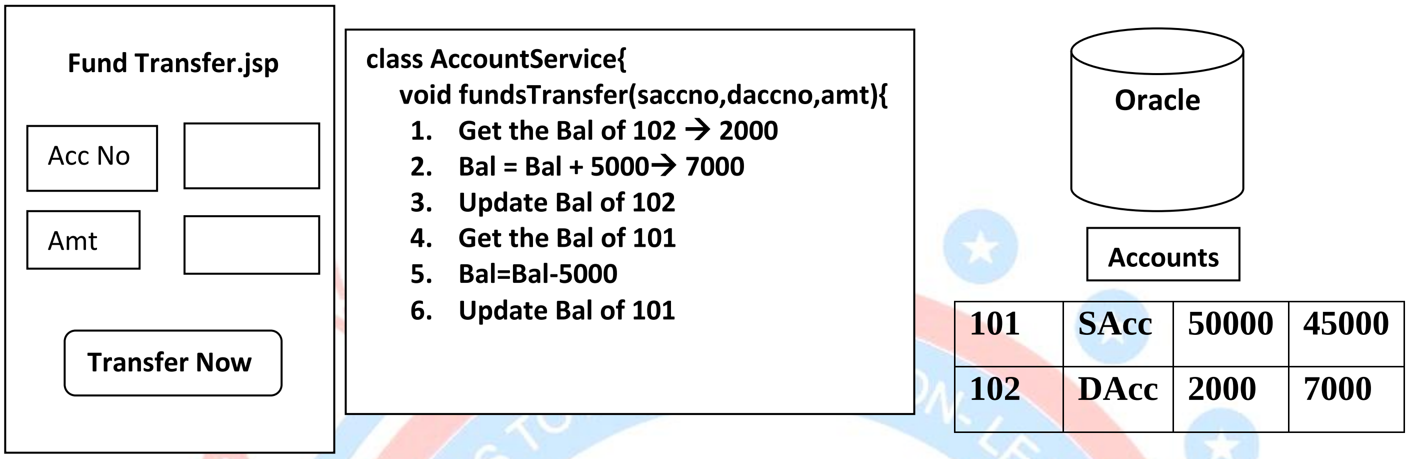
Hibernate-4 Doc.

Master the Content…

Author

Som Prakash Rai

Transaction management



```
class AccountService{
    void fundsTransfer(saccno,daccno,amt){
        1. Get the Bal of 102 → 2000
        2. Bal = Bal + 5000→ 7000
        3. Update Bal of 102
        4. Get the Bal of 101
        5. Bal=Bal-5000
        6. Update Bal of 101
```

101	SAcc	50000	45000
102	DAcc	2000	7000

Problem

- Amount is added to your friend account but not deducted from your account because of not managing the transactions.

Transaction

- Transaction is the process of performing multiple database operations as one atomic unit with all or nothing Criteria i.e.
  - When all the database operations in the unit are successful then Transaction is successful and should be commited.
  - When any one database operation in the unit is failed then Transaction is failed and should be rolled back.
- When you implement transactions properly in your application, It guarantees ACID properties.

  A –Atomicity

  C –Consistency

  I –Isolation

  D –Durability

1. **Atomicity**
   Atomacity =Definition of Transaction
   Consider the salary transfer
   1000 deposits + 1 withdraw =>Totally salary transfer contains 1001 DB Operations.
- When all these 1001 database operations are successful then Transaction is successful and all 1001 DB operations have to be committed.
- When any one DB operation is failed then Transaction is failed and all DB operations have to be rolled back.

2. **Consistency**
   - When you are running the transaction, you many enforce many business rules as per the application requirement.
   - Your application should be consistent when any business rule is failed otherwise you may get some inconsistent results.

Consider the withdraw operation.

Some business rules are

1. Minimum balance is 5k.
2. if (ATM){
   a. Only 5 withdrawals per day.
   b. Limit: 50k per day.

   }

3. if (Branch){
   a. Only 10 withdrawals per day.
   b. Limit: 5L per day.

   }

- When you are implementing the Transaction, you can implement code for business rules inside transaction. When any business rule is failed, the transaction will be forced to rollback.

3. Isolation:

- You can run multiple transactions concurrently. These concurrently running multiple txs should not disturb not disturb other transactions i.e. multiple transactions should run isolately or independently without affecting each other.

Case A:

- Multiple transactions running concurrently are using multiple Account rows of accounts table.
  Customer 1 having Account number 99
  Customer 2 having Account number 88
  Customer 3 having Account number 77
- **At a time, following operations are happening.**
  1. Transaction 1-trsnafer (withdrawal). -Uses 99
  2. Transaction 2 -bank teller (withdrawal or deposit) -Uses 88
  3. Transaction 3 -loan EMI (withdrawal). -Uses 77
- No problems in the case.

Case B:

- Multiple transactions running concurrently are using single Account row of accounts table.
  Customer 1 having Account number 99 and balance 15K
- At a time, following operations are happening.
  1. Transaction 1-trsnafer (withdrawal). -Uses 99
  2. `Transaction 2- bank teller (withdrawal or deposit) -Uses 99
  3. `Transaction 3 -loan EMI (withdrawal). -Uses 99
- **In this case,**
  - 3 Transactions are running concurrently and using single column or row or table which may cause problems.
- The problems coming when multiple transactions running concurrently are called as Transactional concurrency problems.

<u>There are three Transactional Concurrency Problems</u>

1. **Dirty Read problem**

2. **Repeatable Read problem**
3. **Phantom Read problem**

- To avoid these Transactional Concurrency problems, you have to apply one of the following required Transactional Isolation Levels.

1. **READ_UNCOMMITTED**	**1**
2. **READ COMMITTED**	**2**
3. **REPEATABLE READ**	**4**
4. **SERIALIZABLE**	**8**

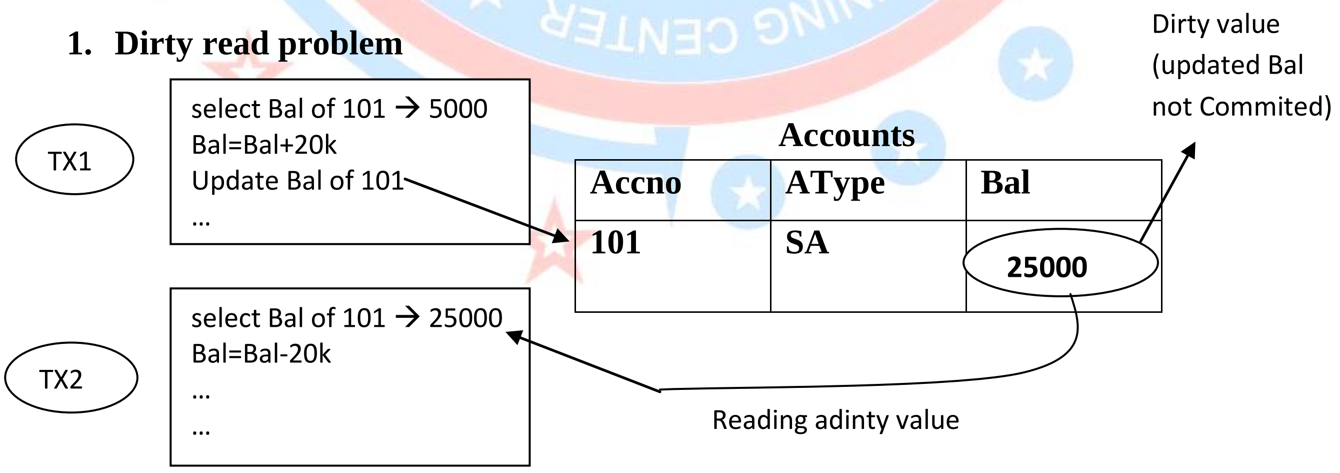
4. **Durability**
   - Your enterprise data should be available for long time i.e. as long as your enterprise application is running.
   - You have to protect your enterprise data from crashes, failures etc.
   - You have to implement proper backup and you can make your enterprise data durable with recovery mechanism and proper logging mechanism.

Consider the following scenario

- Every one minute the enterprise data backup is scheduled.
  - 10.31 A.M First backup
  - Some 1000 DB operations happened
    - You have deposited 10 L at 10.31.57
    - Update accounts set balance =10L where accno =99;
  - 10.31.59 but crashed at this point.
  - 10.32.A.M next backup

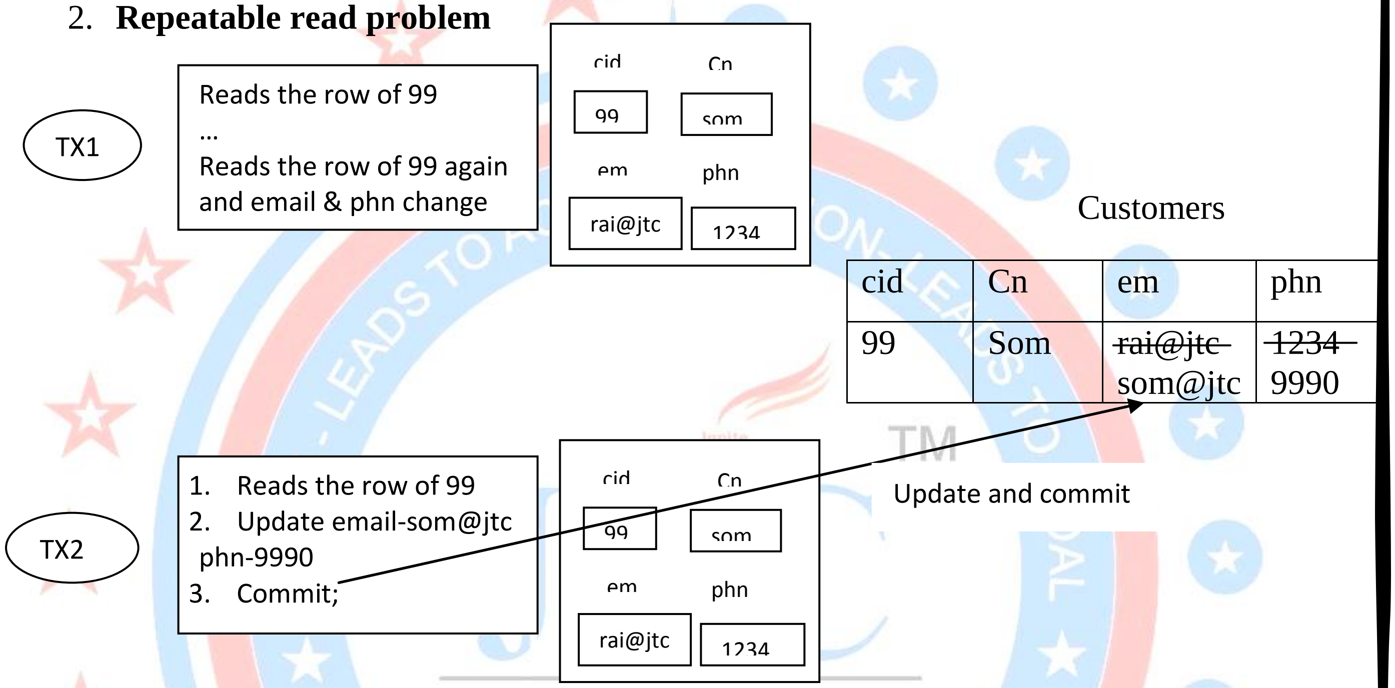
Transactional concurrency problems

1. **Dirty read problem**



- When TRANSACTION reads the Dirty values (i.e. modified but not commited) then you may get some inconsistent results.
- To avoid dirty reads, you have to lock the column (Cell).
- To Lock the column, you have to apply to apply Isolation level called READ COMMITTED.

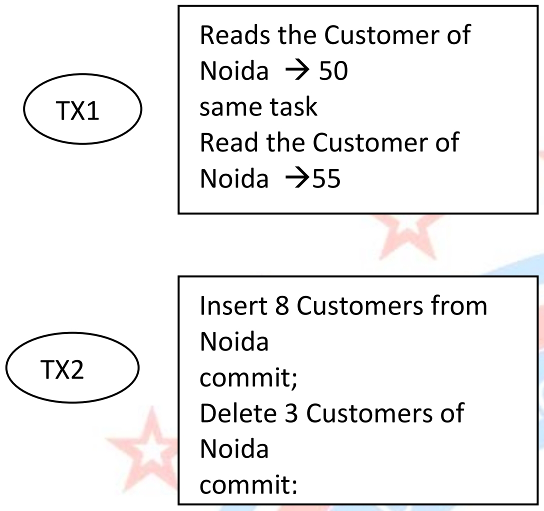
2. **Repeatable read problem**



cid	Cn	em	phn
99	Som	~~rai@jtc~~ som@jtc	~~1234~~ 9990

- When a TRANSACTION is reading the same row repeatedly, you may get different set of values in different reads. This kind of problem is called Repeatable Read Problem.
- To avoid Repeatable Read Problem, you have to lock the row.
- To lock the row, you have to apply Isolation level called REPEATABLE_READ.

3. Phantom Read Problem



Reads the Customer of Noida → 50
same task
Read the Customer of Noida →55

Insert 8 Customers from Noida
commit;
Delete 3 Customers of Noida
commit:

cid	Cn	em	Phn

500 records

50 records (Noida)

55 records of Noida

- When a TRANSACTION is reading the set of rows repeatedly, you may get different set of rows in different reads. This kind of problem is called Phantom Read Problem.
- To avoid Phantom Reads, you have to lock the Entire Table.
- To lock the table, you have to apply Isolation Level called SERIALIZABLE.

	DIRTY READ	**REPEATABLE READ**	**PHANTOM READ**	**LOCK**
READ_ UNCOMITTED	**NO**	**NO**	**NO**	**NO LOCK**
READ_COMMITTED	**YES**	**NO**	**NO**	**COLUMN LOCK**
REPEATABLE_READ	**YES**	**YES**	**NO**	**ROW LOCK**
SERIALIZABLE	**YES**	**YES**	**YES**	**TABLE LOCK**

YES--> Problem Solved NO--> Problem Not Solved

Types of Transactions

1. **Local transactions**
2. **Distributed Transactions**

1. Local Transactions

- When a Single Database is participating in the transactional operations (i.e. DB operations) then it is called as Local Transactions.

EX:

- Transfer the funds from one account to another account where two accounts are in same bank or same database.

1. **Distributed transactions**
   - When a two or more databases are participating in the transactional operations then it in called as Distributed Transactions,
     Ex:
   - Transfer the funds from one account to another account where two accounts are in different banks or different databases.

Types of transactions

1. **Flat transactions**
2. **Nested Transactions**

Flat transactions

Ex:

```
        Begin Tx1
                OP1
                OP2
                OP3
        End Tx1
```

- **Note:** Multiple Flat Transactions running will not disturb other concurrently running transactions

Nested Transactions

Ex:

```
Begin Tx1
            1. OP1
            2. OP2
            3. Begin Tx2
                        OP3
                        OP4
                End Tx2
```

```
    4. OP5
    5. Begin Tx3
              OP6
              OP7
       End Tx3
    6. OP8
End Tx1
```

Note: When Inner transaction (Tx2, Tx3) is failed then outer transaction (Tx1) also will be failed.

Ex: Make a Trip,

Mon-Fri

(Noida>pune>Delhi >Mumbai >Noida)

Book the flights

Tx begin

1. **Book the flight tickets from Noida → Pune(Tx1) -AI**
2. **Book the flight tickets from pune → Delhi(Tx2) -KF**
3. **Book the flight tickets from Delhi → Mumbai(Tx3) -JET**
4. **Book the flight tickets from Mumbai→ Noida(Tx4) -AI**

Tx end.

	JDBC	Hibernate	JPA	EJB	Spring
Local Transaction	YES	YES(JDBC)	YES (JDBC)	YES (JDBC)	YES (JDBC)
Distributed Transaction	NO	YES (CME)	YES (CME)	YES (CME)	YES (CME)
Flat Transaction	YES	YES	YES	YES	YES
Nested Transaction	NO	NO	NO	NO	YES

CME -> Container Managed Environment

Managing Transaction with hibernate

1. **Specifying the Transactional Boundaries**

```
Transaction tx = null;
try{
        …
        tx = session. beginTransaction ();        // TX begin
                OP1;
                OP2;
                OP3;
        tx. commit();        //TX end
} catch (exception e) {
        if(tx!=null){
                tx.rollback();   //Tx end
        }
    }
}
```

2 **Specifying the Isolation Levels**

- Write the following property in hibernate.cfg. xml

  <property name="hibernate.connection.isolation"> 1/2/4/8</property>

Hibernate connection management

- ConnectionProvider is an interface available in org. hibernate.connection package and has the following concrete implementations.
  1. DriverManager ConnectionProvider
  2. C3POConnectionProvider
  3. DatasourceConnectionProvider
- If these built-in ConnectionProviders are not suitable for your requirement, you can write your own ConnectionProvider class by implementing

  org.hibernate.connection.ConnectionProvider interface.

- You can specify the Custom ConnectionProvider in hibernate.cfg.xml as follows.

  <property name="hibernate.connection.provider_class">

```
                com.jtcindia.connection.JTCConnectionProvider
        </property>
```

Built-in ConnectionProvider

A. **DriverManager Connections**
B. **C3P0 Connections**
C. **DataSource Connections**

A) <u>Driver manager connections</u>

- You need to specify the following properties in Hibernate cofigration document.

<u>For MySql database</u>

```
 <property name="hibernate.connection.driver_class">
        com.mysql.jdbc.Driver
</property>
 <property name="hibernate.connection.url">
        jdbc:mysql://localhost:3306/Hibernate7
</property>
<property name="hibernate.connection.username">root</property>
<property name="hibernate.connection.password">somprakash</property>
<property name="hibernate.connection.pool_size">99</property>
 <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
```

<u>For Oracle database</u>

```
<property name="hibernate.connection.driver_class">
        oracle.jdbc.driver.OracleDriver
</property>
<property name="hibernate.connection.url">
        jdbc:oracle:thin:@localhost:1521:XE
</property>
<property name="hibernate.connection.username">system</property>
<property name="hibernate.connection.password">somprakesh</property>
<property name="hibernate.connection.pool_size">99</property>
<property name="dialect">org.hibernate.dialect.OracleDialect</property>
```

B. C3P0 Connections

- C3P0 is Third-Party Connection pooling technique which can be used in any kind of Application
- When you want to use C3P0 Connection, do the following.
  - Add c3p0-0.9.1 jar to project build path.
  - You need to specify the following properties in Hibernate configuration document.

For MySQL database

```
<property name="hibernate.connection.driver_class">
        com.mysql.jdbc.Driver
</property>
<property name="hibernate.connection.url">
        jdbc:mysql://localhost:3306/Hibernate7
</property>
<property name="hibernate.connection.username">root</property>
<property name="hibernate.connection.password">somprakash</property>
<property name="hibernate.c3p0.max_size">99</property>
<property name="hibernate.c3p0.min_size">9</property>
<property name="hibernate.c3p0.timeout">10</property>
```

For Oracle database

```
<property name="hibernate.connection.driver_class">
        oracle.jdbc.driver.OracleDriver
</property>
<property name="hibernate.connection.url">
        jdbc:oracle:thin:@localhost:1521:XE
</property>
<property name="hibernate.connection.username">system</property>
<property name="hibernate.connection.password">mahesh89</property>
<property name="hibernate.c3p0.max_size">99</property>
<property name="hibernate.c3p0.min_size">9</property>
<property name="hibernate.c3p0.timeout">10</property>
```

C. DataSource Connections

1. To use DataSource Connections, Your Hibernate Application must run in CME (Container Managed Environment)

2. You need to specify the following properties in Hibernate configuration document.

For Weblogic Application Server

```
<property name="connection.datasource">MySQLDataSource</property>
<property name="jndi.class">
        weblogic.jndi.WLInitialContextFactory
</property>
<property name="jndi.url">t3://localhost:7001</property>
<property name="connection.username">jtcindia</property>
<property name="password">jtcindia</property>
```

For JBoss Application Server

```
<property name="connection.datasource">MySQLDataSource</property>
<property name="jndi.class">
        org.jnp.interfaces.NamingContextFactory
</property>
<property name="jndi.url">t3://localhost:1099</property>
<property name="connection.username">jtcindia</property>
<property name="password">jtcindia</property>
```

Hibernate Transaction Management

- Transaction is an interface available in org.hibernate package and has the following concrete implementations.
    1. **JDBCTransacrion**
    2. **JTATransaction**
    3. **CMTTransaction**
- TransactionFactory is an interface available in org.hibernate.transaction package and has the following concrete implementations
    1. **JDBCTransacrionFactory**

2. **JTATransactionFactory**
3. **CMTTransactionFactory**

- These TransactionFactory implementation classes are responsible for providing the corresponding Transaction depending on the properties written by you in hibernate.cfg.xml. i.e
    1. **JDBCTransacrionFactory provides the JDBCtransaction**
    2. **JTATransactionFactory provide the JTATransaction**
    3. **CMTTransactionFactory provide the CMTTransaction**

1. JDBC Transactions

- By default, Hibernate uses JDBCTransactionFactory which provides JDBCTransaction. i.e. Default Transaction management used by hibernate system in JDBCTransactions.
- To use JDBCTransactions, Connection Pooling must be

DriverManagerConnectionProvider or C3P0ConnectionProvider

2. JTA Transactions

- When you want to use JTA Transactions, You must check the following.
    - Your Hibernate Application must run in CME (Container Managed Environment).
    - Connections must be Datasource Connections
- To use JTA Transactions, you must specify the following props in hibernate.cfg.xml.

For Weblogic Application Server:

```
<property name="transaction.factory_class">
        org.hibernate.transaction.JTATransactionFactory
</property>
<property name="transaction.manager_lookup_class">
        org.hibernate.transaction.WeblogicTranscationManagerLookup
</property>
<property name="jta.Usertransaction">
        java:comp/UserTransaction
</property>
<property name="hibernate.current_session_context_class">jta</property>
```

3.CMT Transactions:

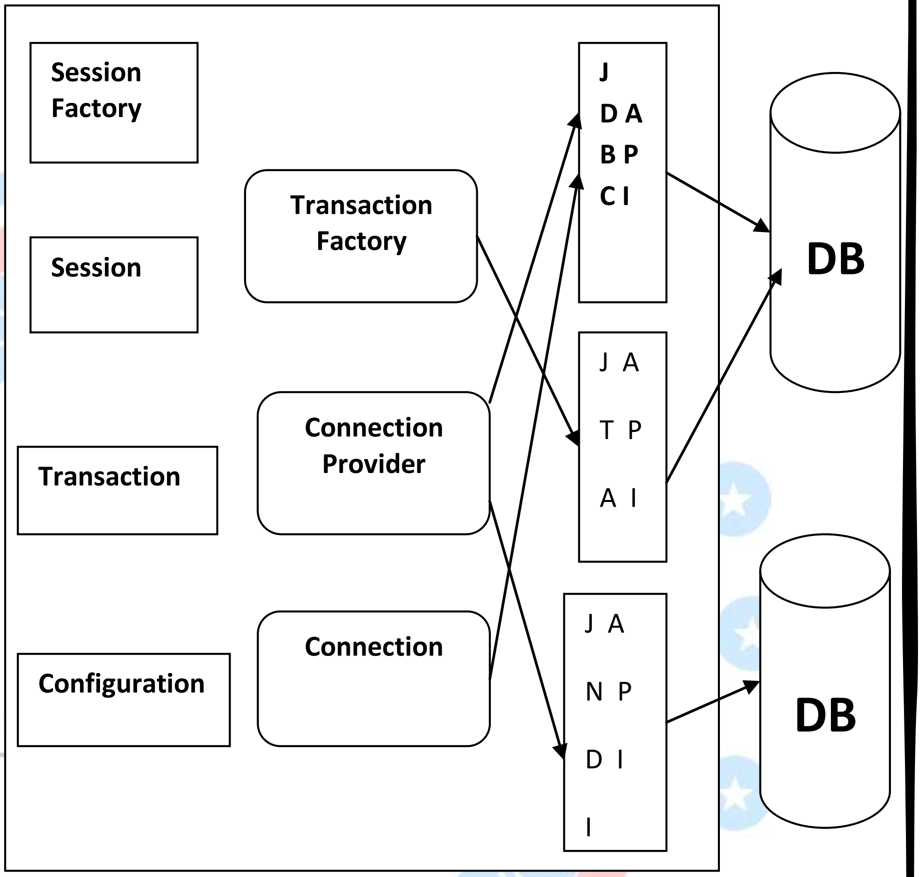
- To Use CMT Transactions, you must check the following.
    - Your Hibernate Application must run in CME(Container Managed Environment).
    - Connections must be Datasource Connections
- To use CMT Transactions, You must specify the following props in hibernate.cfg.xml.

For JBoss Application Server:

```
<property name="transaction.factory_class">
        org.hibernate.transaction.CMTTransactionFactory
</property>
<property name="transaction.manager_lookup_class">
        org.hibernate.transaction.JBossTranscationManagerLookup
</property>
<property name="jta.Usertransaction">
        java:comp/UserTransaction
</property>
<property name="hibernate.current_session_context_class">jta</property>
```

Hibernate architecture

```
try{
        1)cfg=new Configuration()
        2)cfg.configure()
        3) sf=cfg.buildSessionFactory()
        4) sess=sf.openSession();
        5)tx= sess.beginTransaction()
        6) c1 = new Customer (.....)
        7) sess.save (c1)
        8) tx.commit();
        9)sess.close();
}catch(Exception e){
        tx.rollback();
}
```



- When you are working with Hibernate, you are not responsible to write code using JDBC API, JTA API, JNDI API.
- Hibernate Runtime System is Collection of Various Objects running in the system.
- These objects can be devided into two types.

  1. **High Level Objects**
  2. **Low Level Objects**

A. High Level Objects

- Objects used by you in the Hibernate Client Code are called as High Level Objects.

- High Level Objects in Hibernate Runtime System.
  1. **Congiguration**
  2. **SessionFactory**
  3. **Session**
  4. **Transaction**

B. Low Level Objects

- Objects used by High Level Objects internally are called as Low Level Objects.
- Low Level Objects in Hibernate Runtime System.
  1) **Connection**
  2) **ConnectionProvider**
  3) **TransactionFactory**

A. Configuration or AnnotationConfiguration

1. These are available in **org.hibernate.cfg** package.
2. This is the first class that wil be instantiated by the hibernate system.
3. You can use Configuration or AnnotationConfiguration class Object for 2 tasks.
   a. Calling configure() or configure (String) method.
   b. Calling buildSessionFactory () method.
4. Configure () method is responsible for
   a. Identifying Configuration Document.
   b. Reading the data from Hibernate configuration document.
   c. Identifies all the mapping Resources or Mapping classes.
   d. Reading the data from all the Hibernate mapping documents or Annotations specified in Persistence's.
   e. Initializing Configuration object with the data taken from all the XML's or all the Annotations.
5. buildSessionFactory () method is responsible for creating Session Factory object.
6. Configuration or AnnotationConfiguration object is Single Threaded and Short Lived.
7. Once SessionFactory object is created then there is no use of Configuration or AnnotationConfiguration object.

B. SessionFactory

1. This interface is available in org. hibernate package.
2. SessionFactory object is Multi-Threaded and Long Lived.
3. When you call buildSessionFactory () method on Configuration object then following tasks will happen.
   a. Set the defaults to many parameters like Batch Size, Fetch Size, autocommit, etc.
   b. Generates and caches the SQL Queries required.
   c. Generates and executes the Table creation Statements.
   d. Selects the Connection Provider
   e. Select the TransactionFactory
   etc
4. SessionFactory
   a. Factory of Session objects
   b. Client for ConnectionProvider to get the connection.
   c. Client for TransactionFactory to get the transaction.
5. You need to create one SessionFactory per database.
6. When you are using multiple databases then you need to write multiple hibernate configuration documents.
7. **HibernateUtil for one database**

```
package com.jtcindia.hibernate;

import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class HibernateUtil {
        public static SessionFactory factory;

        static {
                Configuration cfg = new Configuration();
                cfg.configure("hibernate-Oracle.cfg.xml");
                factory = cfg.buildSessionFactory();
        }

        public static SessionFactory getSessionFactory() {
                return factory;
```

```
		}
	}
```

8. **HibernateUtil for 2 databases**

Oracle.cfg.xml

```
<hibernate-configuration>
	<session-factory>
		<property name="hibernate.connection.driver_class">
			oracle.jdbc.driver.OracleDriver
		</property>
		<property name="hibernate.connection.url">
			jdbc:oracle:thin:@localhost:1521:XE
		</property>
		<property name="hibernate.connection.username">system</property>
		<property name="hibernate.connection.password">
			somprakash
		</property>
		<property name="dialect">
			org.hibernate.dialect.OracleDialect
		</property>
		<property name="show_sql">true</property>
		<property name="hbm2ddl.auto">update</property>
		<mapping resource="com/jtcindia/hibernate/Student.hbm.xml" />
	</session-factory>
	</hibernate-configuration>
```

MySql.cfg.xml

```
<hibernate-configuration>
	<session-factory>
		<property name="hibernate.connection.driver_class">
			com.mysql.jdbc.Driver
		</property>
		<property name="hibernate.connection.url">
			jdbc:mysql://localhost:3306/hibernate5db
		</property>
		<property name="hubernate.connection.username">root</property>
		<property name="hibernate.connection.password">
```

```
                somprakash
        </property>
        <property name="dialect">
                org.hibernate.dialect.MySQLDialect
        </property>
        <property name="show_sql">true</property>
        <property name="hbm2ddl.auto">update</property>
        <mapping resource="com/jtcindia/hibernate/Student.hbm.xml" />
</session-factory>
</hibernate-configuration>
```

```
package com.jtcindia.hibernate;

import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class HibernateUtil {
        public static SessionFactory oracleFactory=null;
        public static SessionFactory  mysqlFactory=null;

        static {
                Configuration cfg1 = new Configuration();
                cfg1.configure("Oracle.cfg.xml");
                oracleFactory = cfg1.buildSessionFactory();

                Configuration cfg2 = new Configuration();
                cfg2.configure("MySql.cfg.xml");
                mysqlFactory = cfg2.buildSessionFactory();
        }

        public static SessionFactory getSessionFactory(int x) {
                if(x==1){
                        return oracleFactory;
                }else{
                        Return mysqlFactory;
                }
        }
}
```

Usage:
HibernateUtil.getSessionFactory(1) → Gives you SessionFactory to contact Oracle DB.
HibernateUtil.getSessionFactory(0) → Gives you SessionFactory to contact MySQL DB.

C. Session

1. This interface is available in org. hibernate package.
2. Session object is Single Threaded and Short Lived.
3. Session represents period of time where user can do multiple database operations.
4. Session object
   a) Uses transaction factory to get the transaction.
   b) Uses Connection factory to get the Connection.

D. Transaction

1. When Transaction is started, the following tasks will happen:

- Session Cache will be created.
- Connection will be taken and will be associated with the current session.

2. While Transaction is running, following tasks will happen:

- When any Object is participated in Session Operations that will be placed in Session Cache.

3. When Transaction is commited, then following tasks will happen.

- Session will be flushed.
- Session Cache will be destroyed.
- Commit will be issued to database.
- Connection will be released

Types of Object States

- Persistence class Object can be found in 3 states.
   1. **Transient state**
   2. **Persistent state**
   3. **Detached state**

1. **Transient state**
   - When Persistence class Object is newly created and not participated in any session operations then that Object is called as Transient Object.
   - State of that Object is called Transient State.
   - Transient Object does not contain any identity or Primary Key.

2. **Persistent State**
   - When Persistence class Object is participated in any session operations then that Object is called as Persistent Object.
   - State of that Object is called Persistent State.
   - Persistent Object contains any identity or Primary Key.
   - Any modifications happed on the Persistent Object will be reflected to Database.

3. **Detached state**
   - When persistence class Object is participated in any session operations and removed from Session Cache then that Object is called as Detached Object.
   - State of that Object is called Detached State.
   - Detached Object contains any identity or primary key.
   - Any modifications happed on the Detached Object will not be reflected to Database.