**<https://docs.jboss.org/hibernate/orm/4.2/devguide/en-US/html/ch02.html>**

### Hibernate Transactions

A Transaction is a unit of work in which all the operations must be executed or none of them. To understand the importance of transaction, think of an example which applies on all of us i.e. Transferring Amount from one account to another as this operation includes the below two steps:

* Deduct the balance from the sender’s bank account
* Add the amount to the receiver’s bank account

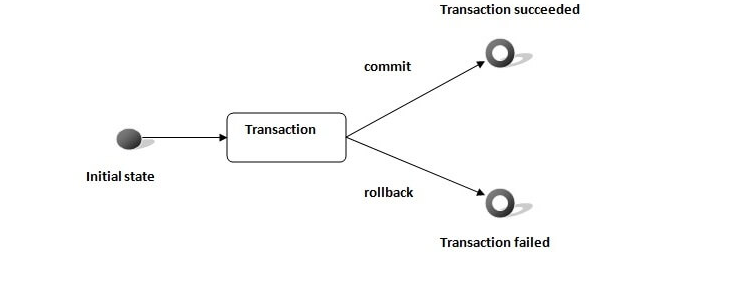


Fig. 2: Lifecycle of an Atomic Unit of Work (i.e. A Transaction)

Now think a situation where the amount is deducted from sender’s account but is not delivered to receiver’s account due to some errors. Such issues are managed by the transaction management where both steps are performed in a single unit. In the case of a failure, the transaction should be roll-backed.

#### **1.2.1 Hibernate Transactions Properties**

Every Transaction follows some transaction properties and these are called as **ACID** properties. ACID stands for Atomicity, Consistency, Isolation, and Durability.



Fig. 3: Transaction Management ACID Properties

* **Atomicity**: Is defined as either all operations can be done or all operation can be undone
* **Consistency**: After a transaction is completed successfully, the data in the datastore should be a reliable data. This reliable data is also called as consistent data
* **Isolation**: If two transactions are going on the same data then one transaction will not disturb the other transaction
* **Durability**: After a transaction is completed, the data in the datastore will be permanent until another transaction is going to be performed on that data

#### **1.2.2 Hibernate Transactions Interface**

In Hibernate framework, we have Transaction interface that defines the unit of work. It maintains the abstraction from the transaction implementation (JTA, JDBC). A Transaction is associated with Hibernate Session and instantiated by calling the sessionObj.beginTransaction(). The methods of Transaction interface are as follows:

| **Name** | **Description** | **Syntax** |
| --- | --- | --- |
| begin() | It starts a new transaction. | public void begin() throws HibernateException |
| commit() | It ends the transaction and flushes the associated session. | public void rollback() throws HibernateException |
| rollback() | It rolls back the current transaction. | public void rollback()throws HibernateException |
| setTimeout(int seconds) | It set the transaction timeout for any transaction started by a subsequent call to begin() on this instance. | public void setTimeout(int seconds) throws HibernateException |
| isActive() | It checks if this transaction is still active or not. | public boolean isActive()throws HibernateException |
| wasRolledBack() | It checks if this transaction roll backed successfully or not. | public boolean wasRolledBack()throws HibernateException |
| wasCommitted() | It checks if this transaction committed successfully or not. | public boolean wasCommitted()throws HibernateException |
| registerSynchronization(Synchronization synchronization) | It registers a user synchronization callback for this transaction. | public boolean registerSynchronization(Synchronization synchronization)throws HibernateException |

#### **1.2.3 Hibernate Transaction Management Basic Structure**

This is the basic structure that Hibernate programs should have, concerning Transaction Handling

|  |  |
| --- | --- |
| 01 | Transaction transObj = null; |
| 02 | Session sessionObj = null; |

|  |  |
| --- | --- |
| 03 | try { |
| 04 | sessionObj = HibernateUtil.buildSessionFactory().openSession(); |

|  |  |
| --- | --- |
| 05 | transObj = sessionObj.beginTransaction(); |
| 06 |  |

|  |  |
| --- | --- |
| 07 | //Perform Some Operation Here |
| 08 | transObj.commit(); |

|  |  |
| --- | --- |
| 09 | } catch (HibernateException exObj) { |
| 10 | if(transObj!=null){ |

|  |  |
| --- | --- |
| 11 | transObj.rollback(); |
| 12 | } |

|  |  |
| --- | --- |
| 13 | exObj.printStackTrace(); |
| 14 | } finally { |

|  |  |
| --- | --- |
| 15 | sessionObj.close(); |
| 16 | } |

Whenever a HibernateException happens we call rollback() method that forces the rollback of the transaction. This means that every operation of that specific transaction that occurred before the exception, will be canceled and the database will return to its state before these operations took place.