1. **What is a session?(Page no:30 )**

* Session is a single – thread, short lived object representing a conversation b/w application and database.
* It wraps a data base connection.
* By defaults it holds 1st level mandatory cache.
* Factory for transaction.

1. **What is Session Factory?**

* It is a thread – safe, immutable object mapped to single database.
* It is factory for session object and client of connection provider.
* It may holds second level cache data which is optional.

1. **What is Session-per-operation anti pattern?**

Session-per-operation is nothing but don’t open and close a session for every simple query with in a thread.

1. **What is a transaction?**
   * It is single – thread, short lived object used by application to represent a unit of work.
   * A session may have multiple transactions.
2. **What is flushing?**

The process of synchronizing the memory state with database at the end of a unit of work.

1. **What is dirty check?**

Hibernate automatically detects weather object has been modified and if updates automatically. This is called automatic dirty checking.

1. **Generators in Hibernate?(**63 page 0f HB reference**)**
   * Org.hibernate.id.IdentifierGenerator.
   * Assigned: default algorithm. Start with 0.
   * Increment: max value + 1. Start with 1.
   * Sequence: not work in my sql. If no sequence is specified it will check for “hibernate\_sequence” having logic to increment +1.
   * Hilo: it takes support of helper table. Allow properties of table, column, and max\_lo. It uses formula (high Val)\*(max\_lo+1). For each identifier value high value increased by 1.
   * Sequence Hilo: same as Hilo but the difference is it takes high Val from sequence. Allowed properties are sequence, max\_lo.
   * Native: it checks weather the underlying database supports identity. If not checks for sequence if not then Hilo algorithm.
   * Foreign:uses the identifier of another object. Usually uses in the case of one – to - one association**.**

<generator class=”foreign”>

<param name=”property”>person</param>

</generator>

* + User defined: over ride generate (Session Implementer, Object) method. Return type is Serializable and this method is available in IdentifierGenerator.

1. **What are Hibernate object states? (121)**
   * **Transient**:
     + An object is transient; it just created by using new keyword and not associated to session and also has no identifier column.
     + This object will be destroyed by garbage collector if application does not hold any reference anymore.
   * **Persistence**:
     + Ttransient object will be made persistence using **Session object**.
     + It has an identity column and associate with a session obj.
     + Hibernate will synchronizes with database if any changes occurs in object after completion unit of work.
   * **Detached**: a detached instance is an object that has been persistent, but its session has been closed or clear or evict. Detached instance can be reattached to a new session.
2. **What is composite primary key? Or composite identity field**
3. A table has more than one primary key then we call it as composite primary key. These are mapped by using <composite-key> tag instead of <id>.

<Composite-key>

<key-property name = “id” column = “emp\_id”/>

<key-property name = “username” column = “username”/>

</composite-key>

1. When table has composite primary key then that corresponding pojo class must implement java. Io. Serializable interface because when we load an object we pass primary key column value but in this case many values are there. So we set these values to pojo object and pass that object as parameter.
2. **What is versioning?**
   * It is the concept of keep tracking, how many times a particular record is modified. Version column value starts with ‘0’. We configure this through <version**>** tag place after the <id> tag.
   * <version name=”name” column=”name”/>.
3. **What is pagination?**

Pagination is the concept that allows us to specify maximum number of records to retrieve and/or first row to retrieve.

Query. setMaxResults (3);

Query. setFirstResult(4);

1. **Difference b/w named parameters and positional parameters?**

Named parameters having below advantages.

* + In – sensitive to the order they occur in the query.
  + They may occur multiple times in query.
  + More readable (self documenting).

1. **What is state less session?**

State less session does not have

* Any Persistent context associated with it.
* First level and second level cache.
* Dirty checking or transaction writes behind.
* StatelessSession session=Session Factory. openStateLessSession();

1. **What is the difference between sorted and ordered collection in hibernate?**   
   **Sorted collection**: A sorted collection is sorting a collection by utilizing the sorting features provided by the Java collections framework. The sorting occurs in the memory of JVM which running Hibernate, after the data being read from database using java comparator. .If your collection is not large, it will be more efficient way to sort it.

**Order collection**: Order collection is sorting a collection by specifying the order-by clause for sorting this collection when retrieval If your collection is very large, it will be more efficient way to sort it.

Inheritance Mapping

Inheritance is the one of the important concept in hibernate. It will save the base class object when derived class object is saved.

1. Table per class hierarchy (discriminator column).
2. Table per sub class
3. Table per concrete

Suppose we have an interface Payment with the implementers CreditCardPayment, CashPayment, and ChequePayment.

## Table per Class Hierarchy:

* Parent class object data will be saved when derived class object is saved into a single table.
* We can use discriminator column to maintain which subclass object is stored.
* If discriminator column is not available hibernate throws Exception.
* Exactly **one table is required**. There is a limitation of this mapping strategy: columns declared by the subclasses, cannot have NOT NULL constraints because we will store null values for other class property columns.

<class name = “Payment” table = “payment”>

<id name = “id” column = “id” generator = “increment”/>

<discriminator column="payment\_type" type="string"/>

<property name="amount" column="amount"/>

<**subclass** name="CreditCardPayment" discriminator-value="credit">

<property name="creditCardType" column=" cctype "/>

</subclass>

<**subclass** name="CashPayment" discriminator-value="cash">

<property name="creditCardType" column="cctype"/

</subclass>

</class>

## Table per sub class:

* X number of classes = x number of tables.
* Maintain a column in derived class tables to join the records with base class table primary key.
* All subclass tables have primary key associations to the super class table so the relational model is actually a one-to-one association
* We use <join-subclass> tag under <class> tag.

<class name="Payment" table="PAYMENT">

<id name="id" type="long" column="PAYMENT\_ID">

<generator class="native"/>

</id>

<property name="amount" column="AMOUNT"/>

<joined-subclass name="CreditCardPayment" table="CREDIT\_PAYMENT">

<**key column**="PAYMENT\_ID"/>

<property name="creditCardType" column="ccTYPE"/>

</joined-subclass>

<joined-subclass name="CashPayment" table="CASH\_PAYMENT">

<**key column**="PAYMENT\_ID"/>

</joined-subclass> ....(rest of class configuration)

</class>

## Table per Concrete class:

* X number of derived classes = x number of tables.
* Once we save the derived class object, then derived class data and base class data will be saved in the derived class related table in the database.
* The limitation of this approach is that if a property is mapped on the super class, the column name must be the same on all subclass tables. The identity generator strategy is not allowed in union subclass inheritance.
* We use <union-subclass> tab under <class> tag.

<class name="Payment">

<id name="id" type="long" column="PAYMENT\_ID">

<generator class="sequence"/>

</id>

<property name="amount" column="AMOUNT"/>

...

<union-subclass name="CreditCardPayment" table="CREDIT\_PAYMENT">

<property name="creditCardType" column="CCTYPE"/>

</union-subclass>

<union-subclass name="CashPayment" table="CASH\_PAYMENT">

</union-subclass>

</class>

Component Mapping

* Component refers object oriented notation of Composition (Has – a).
* Use is reusability.
* For ex we have two classes like Person and Name and both having “Has-a” relation.

public class Person {

private java.util.Date birthday;

private **Name** **name**;

private String key;

//getters and setters methods

}

Public class Name {

char initial;

String first;

String last;

}

* + <class name=”Person” table=”person”>

<id name=”id” column=”id/>

<property name=”age” />

<component name=”name” class=”Name”>

<property name=”initial”/>

<property name=”first name”/>

</component>

</class>

* The person table would have the columns pid, birthday, initial, first and last
* Like all value type, components don’t support shared references. I.e. if two persons have the same name, but the two person objects would contain two independent name objects, only “the same” value.
* The properties of components may be any hibernate type.

Association Mapping

**Uni directional vs. bidirectional**:

* The main difference is that bidirectional relationship provides navigational access in both directions, so that you can access the other side without explicit queries. Also it allows you to apply cascading options to both directions.
* Note that navigational access is not always good, especially for "one-to- -many" and "many-to-very-many" relationships. Imagine a Group that contains thousands of Users:
  + How would you access them? With so many Users, you usually need to apply some filtering and/or pagination, so that you need to execute a query anyway
* There are 4 types of association mappings

1. One – to – one
2. One – to – many
3. Many- to – one
4. Many – to – many.

## One –to –one

* A one-to-one relationship occurs when one entity is related to exactly one occurrence in another entity.
* We can use **foreign generator** in child table mapping file.
* By default lazy loading (fetching the depending class record at a time) is disabled.

<one-to-one name="stockDetail" class=" StockDetail" cascade="save-update">

</one-to-one>

* In child class mapping file

<one-to-one name="stock" class=" stock.Stock" **constrained**="true">

</one-to-one>

* The main difficulty in this **one-to-one** relationship is ensuring both are assigned the same primary key. In StockDetail.hbm.xml, a special **foreign identifier generator** is declared; it will know get the primary key value from STOCK table. With constrained=”true”, it ensure the Stock must exist.

## One – to – many:

* By default lazy loading is true
* We can use either list or set or map.

<set name=*"address"* table=*"address"* cascade=*"all"* >

<key column=*"forgienid"* /> 🡪foreign key in the child table

<one-to-many class=*"pojo. Address” /*>

</set>

* Unidirectional one – to – many on foreign key is very unusual. So we can use join table for this type of association.

<class name="Person">

<id name="id" column="personId">

<generator class="native"/>

</id>

<set name="addresses" table="PersonAddress">

<key column="personId"/>

<many-to-many column="addressId"

Unique="true" class="Address"/>

</set>

</class>

<class name="Address">

<id name="id" column="addressId">

<generator class="native"/>

</id>

</class>

## Many – to – one:

## What is use of inverse attribute in collection?

* Applicable only for one – to – many and many – to – many.
* The keyword inverse*=true* means the related table is relationship owner and *inverse=false* means table is not the relationship owner.
* The *inverse* keyword tells Hibernate which table is the relationship owner between the two tables and the class related to the relationship owner table will UPDATE the relationship.

HQL

* Hibernate Query Language is a fully Object Oriented notation understanding language.
* Hibernate queries are case insensitive except java classes and properties.
* Hibernate converts HQL to SQL by using AST query translator.
* HQL doesn’t support “insert”. INSERT INTO ... SELECT ... form is supported
* In case of sql we use table names and columns name but in hql we use pojo
* Query may returns multiple objects/ properties as
  + Object [].
    - Select p. first name from Person p;
  + Or List
    - Select new List(first name) from Person
  + Or as type safe java Object
    - Select new Person(first name) from person
* HQL supports two types of parameters
  + Named parameters(:p1)
  + Positional parameters (? 0 based index)
  + After named parameter positional parameters are not allowed.

**Named Queries**:

<query name="findStockByStockCode">

<![CDATA[from Stock s where s.stockCode = :stockCode]]>

</query>

* In case of native sql query

<sql-query name="findStockByStockCodeNativeSQL">

<return alias="stock" class="com.mkyong.common.Stock"/>

<![CDATA[select \* from stock s where s.stock\_code = :stockCode]]>

</sql-query>

Criteria

Cache

* Cache is the concept to storing the loaded objects in to memory from data base. The main purpose of cache is when we want to load again the same object hibernate load that in to cache not from date base.
* There are 2 types of hibernate cache
  + First level or session level
  + Second level or session factory or jvm level.
  + Query Cache.

## First level cache:

* Default cache enabled for each session and we can’t disable it.
* We can remove it using two methods i.e.
  + evict (Object name) 🡪removes the particular object from session
  + Clear () 🡪removes all the objects from session.
  + contains() 🡪 to check if an object is present in the hibernate cache or not, if the object is found in cache, it returns true or else it returns false
* Any object cached in a session will not be visible to other sessions and when the session is closed; all the cached objects will also be lost.

### Second level cache:

We can enable this cache pre jvm or session factory level by class – by – class or collection – by – collection.

* EH cache. (Easy Hibernate)
* OS cache. (Open Symphony)
* Swarm cache.
* Tree cache (from jBoss).
* To enable configure this in hibernate configuration file.
  + <property name=”cache. Provider \_ class” >org. Hibernate. Cache. EhcacheProvider</property>

Ehcache.xml

<ehcache>

<defaultCache maxElementsInMemory="100" eternal="false" timeToIdleSeconds="120" timeToLiveSeconds="200" />

<cache name="str.Product"maxElementsInMemory="100" eternal="false" timeToIdleSeconds="5" timeToLiveSeconds="200" />

  </ehcache>

Strategies (usage) of cache:

* Transaction
* Read only
* Non restrict read - write.
* Read – write.

Interceptors

* Interceptor interface provides call-backs form session to application allowing the application to inspects or manipulates the properties of persistence object before it is saved or loaded or updated or deleted.
* One possible use for this is to track auditing information.
* We can implement Interceptor interface or extend (better) Empty Interceptor.
* Interceptor has came in to two flavours
  + Session Factory scoped (new Configuration. set Interceptor (new Our Interceptor ())).
  + Session scoped (sf. open Session (new Our Interceptor ())).

Filter

1. **What is hibernate Filter?**

Filter is a *global, named, parameterized* filter that may enable or disabled for a particular hibernate session used to ***handle data* with visibility rules.**

* Filter is like database views but parameterized inside application.
* <filter – def name=”filter1”>
  + <filter - param – name =”one” type=”int”/>
  + <filter - param – name= “two” type=”float”/>

</filter – def/>

* We can attach filter to a class or collection. For ex,
  + <filter name=”filter1” condition=”:one<10 and two>100”/>
* We can enable or disable the filter on a particular session.
* We can’t use filters on Native SQL queries.