Interview Preparation

# Hibernate

## ORM (Object Relation Mapping)

Is a programming tech to map application domain model to relation DB table and vice-versa

Is a persistence framework. Used to implement persistence operation or DB Operation.

Use hibernate persistence API for CRUD Operation.

Provide option to map POJO object to traditional DB tables with the help of XML or Annotation configuration.

## ORM Advantage

Eliminate all boiler plates code that come with JDBC & take care for managing resources  
. Support XML and annotation.  
. Provide HQL, Criteria, NativeSQL, NamedSQL  
. Provide lazy initialization using proxy object & perform actual DB queries only when its required.  
. Provide various caching mechanism, that helps to improve appn performance.

. Support inheritance, association and collection.

. Provide Tx management.

. JDBC throws SQL Exception that is checked Exception But Hibernate throws HibernateException.

## ORM Feature

Hibernate is responsible for taking connection, create statement and release the resources.

Generate value required for PK column.

In-Build Primary Key algo and support custom implementation also.

Support various mapping style (Simple, Collection, Inheritance, Association mapping)

**. Inheritance**  
 Table per class mapping

Table per sub-class Mapping

Table per concreate class mapping

**Association Mapping**

One to one

One to many  
 Many to many  
Have build in support for batch.

Having various caching mechanism.

Various object-oriented query Language HQL, QBC, Native, NamedSQL

## openSession Method

Always open a new session, once done we should close this session object. We should open a new session for each request in multi-threaded environment.

## getCurrentSession Method

Return session object bounded with context. We have to modify in cfg file for this to work. Since this session object is belongs to hibernate context, we don’t need to close it Once SF is closed the object get closed.  
**<property name = “hibernate. current-session-object-class”> thread</property>**

Hibernate session objects are not thread safe. So, we should not use it in multi-threaded environment.

## openStateless Session

Return instance of stateless session object. This method doesn’t implement first or session level cache, Tx, Collection or any event model & interceptors.

It’s more like normal JDBC connection that doesn’t provide any benefits that comes with hibernate framework.

Ex: Loading bulk data & doesn’t want hibernate session to hold data for caching

## Hibernate Proxy

Hibernate use proxy class for loading data only when it’s required. This is done by extending entity beans. If beans are final then lazy loading is not possible, hence low performance.

## Hibernate Fetch (Eager and Lazy)

By default, hibernate use Lazy loading

**@ElementCollection (fetch=FetchType.EAGER)**

**@ElementCollection (fetch=FetchType.LAZY)**

## Save Method

Can be used to save entity in DB, can invoke without Tx

If used without Tx & having cascading between entities, then only primary entity get saved unless we flush the session.

Save method returns generated id immediate, this is possible bcoz primary object is saved as soon as method invoked. We should avoid saving outside Tx boundaries otherwise mapping entities will not save cause data inconsistency problem.

## Persist Method

Hibernate persist () is same as save with Tx & adds the entities object in the persistent context. So, any further changes are tracked.

If any object properties are changed before Tx is commit or session is flushed. It will also be saved in DB.

We can use persist () only with Tx

Doesn’t return anything, so we need to use the persistence object to get the generated identifier value.

## saveOrUpdate Method

it returns result into insert or update queries.

Based on the provided value, If the data is present in DB update query will execute.

We can use this method without Tx also, but again we face same issue with mapping object are not getting saved, If session is not flushed.

## Update Method

Should be used only for updating the entities information. This method add the entities into persistence context & any further change are tracked & saved when Tx commit.

## Update & Merge Method

Suppose we are creating a session and load an employee’s object, now emp obj is in session cache, If we close the session at this point & update (), it will throw Exception.

To make other persist we need to open another session. Now we load same object again in current session.

So, if we want to update present object with previous changes, we have to use merge ()

Merge () will merge changes of both state of object & will store in DB.  
If you want to save ur modification at anytime without knowing about the object state, Then use merge()

## Hbm2ddl

hbm2ddl option  
-. create: create schema, delete perv data

. update: update schema  
. create-drop: Drop schema when SF is closed.

. validate: validate the schema

## Type of Object State

**Transaction State**: When persistence class object is newly created and not participation in session.

State is Transaction State and Object is Transaction Object.

Doesn’t contain Primary Key

**Persistence State**: When object is participating in session, have primary key or Identity key.

Any modification on persistence object reflects in the DB

**Detached State**: When object is removed from session, contain primary key or Identity key.

Any modification on this object doesn’t reflect in DB.

Hibernate only check for Persistence state (object)

## Hibernate Methods or API used

@Entity –

@Table (name = “account”)

@Id -- @GeneratedValue (Strategy = GenerateType. auto)

@Column (name = “”)

@CollectonofElements / @ElementCollection

@JoinTable (name=””, joincolumns = “”) --- @JoinColumn(name=””)

@indexCoumn -- @lob

@OnetoOne -- @ManytoOne

@Transient -- @Temporal (TempralType --- > Date, Time)

@JoinTable

@Embeded -> clue for hibernate where I am using

@Embeddable -> if doesn’t want to create separate table

@Cascade (value = CascadeType -> ALL, NONE, Persist, save and many more)

@Inhertance (strategy = InheritanceType.SINGLE\_TABLE, InheritanceType.JOINED)

@primaryKeyJoinColumn (name = “sid”)

## @Embeddable vs @Embedded

Embeddable - > To mark class Embeddable, if u don’t want to create separate class.

Embedded - > Clue to Hibernate where I am using

## Hibernate Interface

Configuration / AnnotationConfigration (C): org.hibernate.cfg. package.

single thread and short lived.

First class initialized by hibernate appn by calling

Configure () – () -> Default

Configure () – () -> String

Configure () is responsible for - > identify cfg doc/read data from cfg file.

Initialize configuration object by reading mapping/resource from cfg file.

buildSessionFactory () is responsible for creating session factory object.

Once SF object is created then there is no use of configuration object.

SessionFacory (I): org.hibernate

Multithread & long lived

Will call buildSessionFactory () on configuration object.

Task performed by buildSessionFactory

Set default to many parameters like batch, fetchsize, autocommit

Generate & cache SQL queries if required.

Generate execute table creatin statement.

Take connection provider/Tx Provider

SessionFacory in the factory of session object

only one SF per DB

Session (I) org. hibernate

Single thread & sort lived

Session represents period of time where user perform multiple DB operations

Session objects use TransactionFactorynto get Tx.

Session objects use connection from connection provider

Tranasaction (I) org. hibernate

When started: session cache will be created, Connection will be taken & associated with current session.

When Tx is running: If any object is participated in session, then that is placed in session cache

When Tx commit: session flushed, session cache is destroyed, committed to DB, Release Connection.

## Tx Management

Tx is the process of performing multiple Db operation as one atomic unit with All or nothing

ACID:

A (Atomicity)—All the changes are performed or none

C (Consistency) – Data is in constant state when Tx Start & End

I (Isolation) – Multiple DB operation should run independently. They should not affect others

D (Durability) –Once changes done, those are saved permanently in DB

Tx Type: Local and Distributed

Local: -Single DB participation

Distributed: Multiple DB participated

Specify Isolation Level

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

getDefaultTransactionIsolation () for DB

## Isolation level

Read Uncommitted (1)

Lock cell --- Read Committed (2)

Lock Column / row --- Repeatable Read (4)

Lock Table – Serializable (8)

Problem – Dirty Read Problem

Repeatable Read Problem

Phantom problem

## Hibernate Fetching

To optimize hibernate generated select statement

. Fetch: fetch-join, disable Lazy loading, always load all the collection and entity

-select(default): lazy, load all the collection and entities

-batch-size N: fetch upto N collection or entity

-fetch-subselect: Enable all its related collection in a sub select statement

@Fetch (FetchMode -- subselect, join, select)

@BatchSize (size = 10)

## Hibernate Proxy

A proxy object is just a way to avoid retrieving an object utill we need it.

Lazy --- By Default

----fetch data when u want

---- select fetching strategic (optimize memory)

Eager -> fetch immediately (optimize time)

Fetch-join -- > strategic

@ElementCollection (fetch = FetchType. Lazy or FetchType.Eager)

## Session Load Method

Reattaching object without synchronize or update with DB, By the way u can use

session. update () to sync with DB during re-attachment.

## Get vs load method

**Get** -- load data as soon as it called (Eager)

**Load** – return proxy object & load data only when its needed, used Lazy loading

Throws Exception, when data is not found, we should use it only when we know data exist.

**Get** – When we want to make sure data exist in DB (return null if no data)

**Load** –Exception (objectNotFound) R.E

**Get** – When we call session.get (), it hit Db & return the object, this id the reason it generates the query, when we update any value of the object.

In this case object of given id doesn’t exist in DB, then it will return NULL, instead of throwing Exception

## Hibernate Cache

Is used to improve appn performance, The idea behind is to reduce number of DB queries

**First Level Cache**: Associated with session object, it’s enable by default & there is no way to disable it.

However, hibernate provide some method’s using that we can delete selected object from cache or clear the cache completely.

Any object cached in the session is not visible to other session.

When session is closed all the cached object will be lost.

First level is session specific, that’s y we get the same object & no queries triggered. Whereas in other session queries is fired to load data.

First level cache has old data, any update will be reflected from next session

Can use session evict () to remove a single object.

Session clear (), to clear the cache (delete all object from cache)

Session contains, to check if any object is present in hibernate cache or not. If found true else false.

**Second Level Cache:** is maintained by the Session Factory & available for everyone, but 1st Level cache is only for session. Hibernate always tries to get 1st level cache from session, if fails tries with 2nd level cache. Then it hit DB

Hibernate static initialize (), which populate proxy object will attempt to hit 2nd level cache before going to DB.

**Query Cache:** Query cache, store the result of SQL query for future call, Query cache is used with 2nd level cache to improve performance.

## Concurrency Strategic

This is mediator which is responsible for storing and getting data from cache

Transaction ---- for read-mostly data

Read-write

Nonstrict-read-write

Read-only

Transaction

## Cascade

When we have relationship b/w entities, then we have to define how operation will affect another entity. This we control via cascade.

**NONE**: No cascade, no operations in parent will affect child.

**ALL**: cascade, save, update, delete, evict, lock, merge, basically everything.

**Save-update:** cascade save-update (only hibernate)

Persist: for save or persist operation

Merge, Refresh, remove: remove all related entities associated with

Detach: Detach all the related entities if a manual detach occurs

**Orphan Record:**  An orphan record is a record in a child table, but it’s doesn’t associate with parent.

In appn, if child record is removed from the collection & if we want to remove the child record immediately from the DB, then we should use cascade = “all-delete-orphan”

## HQL

session.createQuery (“Query String”).list().setParameter(0,1001)

## Criteria

session.createCriteria (Account.class)

.add(Restriction.eq (“name”,” a”))

.add(Restriction.gt (“balance”,” 12444L”)) .list()