Hibernate interview questions.

------------------------------------

1. what is diff between jdbc and hibernate. and why we are used hibernate.

-------------------------------------------------------------------------------

JDBC

--------

-> it is database dependent.

-> jdbc does not send the data in the form of object. it will allow to send the data in form of text or value.

-> jdbc is an api provided by third party vendor. its throw sql exception i.e checked exception so we need to write lot of try catch block.its a boiler plate code.

-> jdbc does not support cache mechanism so performance of application is low.

-> jdbc code is very tightly coupled with the application.

-> jdbc does not support relationships.

Hibernate

--------------

-> hibernate is a database independent so we can run any platform.

-> but in real-time we need to transfer the data in the form of objs. in hibernate we can send the data in the form of object.

-> hibernate is an ORM framework. its throw uncheked exception so we no need to write try catch an throws exception.hibernate builts in transaction managemnet

system.it removes the usage of try catch blocks.

-> hibernate support caching mechanism .it improves the performance of the application for efficient data retrieval.

-> hibernate support jpa annotaions. so it removes lot of boiler plate code that comes jdbc api the code looks like more readable and clears.

-> its support inheritance or association relationship.

2. what is ORM (object relational mapping).

------------------------------------------------

-> Hibernate is the most-popular persistence framework and ORM tool for Java Applications.

-> ORM (Object/Relational Mapping) is a methodology where objects in Java Applications are persisted transparently in the relational database tables.

-> ORM stands for Object Relational mapping. It is programming paradigm which is used to persist java objects to database tables.

3. what is session.

-----------------------------

-> session is an interface present in "org.hibernate.package".

-> it is used to perform db operation. session provide methods to perform create, read, update, and delete operations for a persistent obj.

-> we can execute hql quries, sql native quires and create criteria using session obj.

4. what is Session Factory.

-----------------------------------

-> session factory is an interface present in "org.hibernate package".

-> session factory is a state which contain mapping file & configuration file information and create the connection to perform the db operation.

-> Session Factory is a heavy weight object and it should be created one per database. SessionFactory object is shared by multiple sessions.

-> session factory is a heavy weight obj that has to be created only once per application.

-> when u have multiple db in your application you should create multiple session factory objects. eg if u are using two db called mysql & oracle in your

hibernate application then u need to build 2 session factory objects.

sessionfactory factory = newconfiguration().buildsessionfactory();.

5. what is diff between session and session factory.

---------------------------------------------------------

-> session

--------------

-> session is not a threadsafe.

-> it is mutable.

-> it is a lightweight.

-> performance wise it is high compare to session factory.

sessionfactory

--------------------------

-> sessionfactory is threadsafe so, many thread can access it concurrently or simulteniously.

-> it is immutable and it will be created as singleton while the server initialize itself.

-> it is heavy weight becoz it maintain datasource, mapping file, and configuration file information.

-> performance wise it is low compare to session.

why should we need to make sessionfactory obj of hibernate as singleton.

----------------------------------------------------------------------------------

-> in hibernate a sessionfactory obj is the only one heavy obj becoz it stores configuration data and all the mapping data of a project.

-> if our project is a desktop application, then there is no need to make sessionfactory obj as a singleton, becoz only one user can access the

application at a time.

-> if a proj is a distributed application, it means it is a web application, remoting application, then at a time multiple clients can send a request

to the server application. if multiple sessionfactory objets are creted then the burden on the server will be increased. so we should make session

factory obj as a singletone.

-> by default sessionfactory obj of hibernate is not a singleton, it is the programmer responsibility to make it as a singleton.

public class HibernateUtil{

private static SessionFactory factory;

private HibernateUtil()

{}

public static synchronized SessionFactory getSessionFactory{

if(factory==null)

{

factory=new configuration().configure("hibernate.cfg.xml").buildSessionFactory();

}

return factory;

}

}

6. what is dialect in hibernate. and how it works.

------------------------------------------------------------

-> dialect is a simple java class present in org.hibernate.dialect package.

-> every configuration file contains dialect which is very important while performing the db operation.

-> dialect class is used to convert hql queries into db specific queries.

-> if u want to shift from one db to another db just change the dialect class name and connection details in hibernate cfg file.

Lazy loading & Eager loading.

---------------------------------------

-> it is also an important feature of hibernate which improves performance of the application by reducing network round trips between java application and database .

-> in lazy loading hibernate doesnt load an object from database immediately. an object will be loaded on demand basis.

-> if an object is immediately loaded then that obj is called as eager/early loading.

-> hibernate lazy loading is true by default.

-> since hibernate 3 lazy loading is by deafult enabled so that child objs are not loaded when parent is loaded.

Fetch=FetchType.LAZY.

-> in lazy loading, hibernate creates a proxy object returns proxy obj to the appicaion.

-> when application is accessing the proxy object, then internally the data is selected from the database.

-> with lazy loading feature, no of trips with the database are going to be reduced and performance will be improved.

Lazy loading means when you load parent obj, child objects won’t get loaded until requested.

what is Caching in hibernate.

---------------------------------------

-> it is the most important features of hibernate.

-> caching is a mechanism to store the object which are loaded from the databse.

-> Caching is facility provided by ORM frameworks which help users to get fast running web application.

-> Hibernate also provide this caching functionality, in two layers.

the main advantage of caching is

--------------------------------------

-> it is reduce/save the number of round trips between java application and the database and it improves the performance of an application.

-> when an application wants an object from database, then hibernate looks for that object at level 1 cache, if not found then it looks for that obj at level2 cache.

even though if it is not found, then only it goes for database.

-> hibernate maintains cache at two levels hence hibernate gives better performance.

-> cache is a temperory(buffer) memory that is allocated to RAM.

-> in hibernate cache is managed by two objects.

1.> session object

2.> session factory object.

-> a cache managed by session obj is called as level 1 cache or local cache.

-> a cache managed by session factory obj is level 2 cache.

7. what is first level cache .

------------------------------------------

-> first level cache is automatically created by hibernate when a session is opened and it will be automatically closed when a session will be closed.

-> by default, for each hibernate application the first level cache is bydefault enabled and as a programmer we are not able to enable or disable first level cache.

-> first level cache is associated with "session" object and other session object in application can not see it.

-> first level of cache has a session scope.

-> first level cache session is not sharable with other session, so it is called local cache.

-> as a programmer we don't need to add any special tags in .hbm file or in .cfg file.

-> first level cache is not in the hands of a programmer. it is in hands of hibernate only.

-> each session will have its own cache. a session cannot read an obj from another session cache.

8. what is second level cache.

---------------------------------------

-> second level cache is introduced in hibernate 3.0 and it is associated with session factory object.

-> and its scope is application level. its called global cache.

-> second level cache is created in session factory scope and it is sharable with other session, so it is called global cache.

-> when we are loading an object from the db, then hibernate will search for the obj in the first level cache, if exist then it is loaded from first level cache.

-> if not exist, then hibernate will search for second level cache if exist then the object will be copied to first level cache .

-> there are multiple second level cache s/w are avialable but widely used second level cache is ehcache(EasyHibernateCache), os cache, jboss cache.

-> second level cache is not by default enabled, we need to enabled by adding following two properties hibernate cfg.xml file.

-> add provider class in configuration file.

<property name:"hibernate.cache.providers\_class>

org.hibernate.cache.EHcacheProvider

</property>

-> add cache tag in mapping of pojo class.

<cache usage="read-only"> or read-write value.

-> create a xml file called ehcache.xml and place in classpath.

ehcache.xml it contains.

1.> behaviour of cache

2.> time setting.

3.> update setting.

4.> life time of pojo class

1>. cache:use\_second\_level\_cache.

2>. cache:provider\_class.

-> in order to do setting for a second level cache, we need to create ehcache.xml along with hibernate jars, we need to add a third party jar ehcache-1.2.3 jar.

-> hibernate has provided, provider classes to get the cache from third party cache providers.

1> org.hibernate.cache.ehcache.providers.

2> org.hibernate.cache.oscache.providers.

-> while performing second level cache we should also choose caching strtegy they are.

1.> read only.

2.> read/write.

3.> non strict read/write.

-> we should know different strategies for caching an object.

-> 1.> Read Only: This caching strategy should be used for persistent objects that will always read but never updated. It’s good for reading and

caching application configuration and other static data that are never updated. This is the simplest strategy with best performance because

there is no overload to check if the object is updated in database or not.

-> 2.> Read Write: It’s good for persistent objects that can be updated by the hibernate application. However if the data is updated either through

backend or other applications, then there is no way hibernate will know about it and data might be stale. So while using this strategy,

make sure you are using Hibernate API for updating the data.

configuring of ehcache

--------------------------

-> in hibernate cfg.xml, we need to configure following properties.

<property name= "cache.provider\_class"> org.hibernate.cache.ehcacheprovider </property>

<property name= "cache.use\_secondlevel\_cache"> true </propery>

ehcache.xml

-----------------

<ehcache>

<default cache maxElementMemory="100" eternal="false" timeToIdleSeconds="120"timeToLiveseconds="200"/>

<cache name="Employee" maxElementInMemory="10" eternal="false"timeToIdleSeconds="8" TimeToLiveSeconds="300"/>

<ehcache>

-> in the above xml IdleSeconds indicates waiting time before an object is going to be removed from cache.

-> LiveSeconds indicates a complete lifetime of an object, before it is going to be deleted from cache.

-> to make IdleSeconds and LiveSeconds as working, we need to disable eternal by using eternal="false".

-> in ehcache.xml,if eternal="true" then we should not write timeToIdleSeconds,TimeToLiveSeconds, hibernate will take care by about those values.

-> so,if u want to give values manually better use eternal="false" always, so that we can assign values into timeToIdleSeconds,TimeToLiveSeconds manually.

9. what is diff between first level cache and second level cache.

----------------------------------------------------------------------------

-> first level cache

-----------------------------

-> first level cache is associated with session obj.

-> first level cache is by default enabled.

-> first level cache scope is session level.

-> first level cache is not sharable.

-> first level cache is avilable only untill the session is opened, once the session is closed. the first level cache is destroyed.

second level cache

--------------------------------

-> second level cache is associated with session factory obj.

-> it is optional. if u want then enabled hibernate.cfg.xml.

-> second level cache scope is application level.

-> second level cache is sharable.

-> second level cache is avilable through the application life cycle. it is only destroy when your application is restart.

what is mapping file how to write this.

------------------------------------------------

-> mapping file is a mechanism of placing an object properties into column of a table.

-> a java application can have multiple classes and a database can have multiple tables.

-> hibernate knows how to store an object and how to read it, but it doesn't know which java class object need to be persisted in which table of database.

-> it is the programmer responsbility to tell hibernate that which java class obj need to be stored in which table. to pass this information to hibernate ,

as a programmer we need to construct the mapping files. we can map multiple classes in a single hbm file.

-> mapping can be done by using annotation also. if we use annotations for mapping then we no need to write mapping file. from hibernate 3.0 version its support.

-> each hibernate mapping file must contain one <id> tag property.

-> in mapping file class names & property name are case sensetive.

-> but table name and column name are not case sensative.

-> we can map multiple classes in a single hbm file.

<hibernate-mapping>

<class name="Employee" table="emp">

<id name="emp\_no" column="empno"/>

<property name="emp-name" column="ename"/>

<property name="emp-sal" column="esal"/>

<property name="dept-number" column="deptno"/>

</class>

<hibernate-mapping>

what is configuration file and how to write this.

------------------------------------------------------

-> It is used to bootstrap hibernate and it is used to locate to hibernate mapping file.

-> configuration file is an xml file, this configuration file contains 3 types of information.

1> connection properties.

2> hibernate properties.

3> mapping resource (file name).

-> we can create one configuration file for each database. it means the number of configuration files in a project depends on number of databases.

-> suppose if we want to connect with 2 database like oracle, mysql then we must create 2 configuration file.

-> in hibernate, it internally opens a connection with the database and also it closes automatically.

-> as a programmer we need to provide connection properties to hibernate through configuration file,

<hibernate-configuration>

<session-factory>

<property name="connection.driver\_class">driver class name </property>

<property name="connection.url"> url</property>

<property name="connection.username">username</property>

<property name="connection.password>password</property>

<propert name="show-sql">true</property>

<mapping resource="Employee.hbm.xml/>

</session-factory>

</hibernate-configuration>

10. what is cascade in hibernate.

-----------------------------------------

-> cascade is an attribute. it is a mandatory when we apply parent child relationship between the objects.

-> when we have relationship between entities, then we need to define how the diff operations will affect the other entity.

-> this is done by cascading and there are diff types of it.

-> cascade attributes takes multiple values none, all, save-update, all-delete-orphan.

-> whenever we perform any operation on parent table it perform the operation on child table also.

-> by default values of cascade="none" means no operation will tranfer to the child class.

-> eg if we apply insert(update or delete) operation on parent class obj, then child class obj will also be stored into the database.

-> eg if we apply insert(update or delete) on parent class object will not be effected,if cascade="none".

import.hibernate.annotations.cascade;

@Entity

@Table(name="EMPLOYEE")

public class Employee {

@OneToOne(mappedBy="employee")

@Cascade(value=org.hibernate.annotations.CascadeType.ALL)

private Address address;

}

what is orphan record

--------------------------------

-> it is a record in child table but it doesnt have association with its parent in the application.

-> in an application, if a child record is removed from the collection and if we want to remove that child record immediately from the database, then we need

to set the cascade="all-delete-orphan".

11. what is inverse in hibernate.

-----------------------------------------

-> default value of inverse="false".

-> if inverse="false" hibernate will not check the bi-directional relationships between the tables. in this scenario if we try to send multiple insert queries,

and update queries.

-> if inverse="false", parent class is responsible for saving/updating the child and its relationships.

-> if the inverse is="true" and associated subclass is responsible for saving/updating itself.

NOTE

---------

-> An inverse keyword is always used with the one-to-many and many-to-many relationships.

-> it doesnt work with many to one relationships.

12. what is lifecycle of hibernate or state of an object.

------------------------------------------------------------------------

1. transient state.

-----------------------------

-> in programming world every object has life cycle after birth and before death.

-> there are three states in hibernate obj will perform the persistency.

-> when obj has been created then that state is called as transient state.

-> if we perform any operation also it will not effect into the database or session cache.

-> If object is in transient state, it means it was never associated with session and just created.

2. persistent state.

-----------------------

-> an object is said to be a persistent state, when it is associated with session as well as obj present in database to perform operation.

-> When an object is saved to database using hibernate session via Session.save() or Session.persist() method, then this object is called to be in persistent state.

-> there are several method provided by hibernate to get obj into persistency state eg save, persist, saveorupdate.

3. Detached state.

-------------------------

-> an obj is said to be detached state, when the obj is not associated with session but present in the database.

-> there are several method which can easily detached the obj from persistency state to detached state. eg-close, evict, clear.

-> If object is in detached state, it means session is closed and object is no more part of session.

-> If you call merge or update, object goes back to persistent state.

13. what is HQL.(hibernate Query language).

-------------------------------------------------

-> by calling save(), update(), delete() we can perform bulk operation.

-> we can perform CURD operation on a single obj at a time. if we want to perform CURD operations on multiple objects at a time, we use bulk operation

technique of hibernate at a time. ex- HQL, criteria, Native Sql.

-> hibernate has introduce its own query language with the name of HQL.

-> Hql is a database independent query language. it means we no need of changing a query while connecting with another database.

-> HQL is an Object-Oriented Query language, simillar to database SQL. But instead of working with tables and columns, HQL uses Java Objects to manipulate queries.

-> hql doesn't provide any burden on the developer, becoz its looks like simillar to sql only. so it is easy to learn.

-> a diff between sql and hql is, sql commands are database dependent but hql commands are database independent.

-> to construct Hql queries, we use variable names in place of column names and class names in places of table names.

-> we call hql as object oriented form of sql.

-> HQL is an Object-Oriented Query language, simillar to database SQL. But instead of working with tables and columns, HQL uses Java Objects to manipulate queries.

-> HQL supports:

– FROM Clause.

– AS Clause.

– SELECT Clause.

– WHERE Clause.

– ORDER BY Clause.

– GROUP BY Clause.

– Using Named Paramters.

– UPDATE Clause.

– DELETE Clause.

– INSERT Clause.

– Aggregate Methods.

– Pagination using Query.

-> using hql, we can perform both select and non-select operations on database.

eg. sql > select \*from emp

hql > from Employee e

-> in hibernate, reading a complete row is called reading a complete entity.

-> to load a complete entity, we need to begin hql command with from keyword.

ex. sql > select empno, sal from emp

hql > select e.employeeNumber,

e.employeeSalary from Employee e

-> reading the values of a specific columns is called reading partial entity.

-> in hibernate, to read a partial entity, a query begins with select keyword.

-> if we want to execute hql command, first we need a query object.

ex 1. Query query=session.createQuery("from Employee e");

List list=query.list();

Iterator it=list.iterator();

while(it.hasNext())

{

Employee e=(Employee)it.next();

}

ex 2. Query query= session.createQuery("select e.employeeNumber, e.employeeSalary from Employeen e");

List list=query.list();

Iterator it=list.iterator();

while(it.hasNext())

{

Object[] obj=(Object[]it.next();

sopln(obj[1]);

}

Hql for non-select operaions

-------------------------------------

-> update and delete operations of hql are simillar to sql, but insert operation is different.

-> in hql, insert operation is used to copy the records from one table to another table.

ex. sql > update emp set sal = 9000 where deptno= 10.

Hql > update Employee e set e.employeeSalary=9000 where e.deptNumber=10.

ex.2

sql> delete from emp where sal>1000

hql> delete from employee e where e.employeeSalary >1000.

what is Criteria in hibernate.

---------------------------------------

-> It is an alternative to HQL, it is very useful for the search query involving the multiple conditions.

-> we cannot use criteria to run update or delete quries or any ddl statemnts. its only used to fetch the results from the database using more obj oriented approch.

-> we can read the same output from a database by executing the diff sql commands.

for eg. select \*from emp;

select empno, ename, esal from emp;

-> the above two sql commands will return same output. but second query gives better performance.

-> while reading the data from the database, tuned queries are important to improve the performance.

-> as a java developer, creating tuned queries will increase the burden on the developer.

-> to decrease the burden hibernate has provided criteria API, it internally creates tuned queries and execute them on the database. so a developer is no

need to prepare tuned quries explicitly.

Diff between getOpen session and getCurrent session.

-------------------------------------------------------------

-> getOpen session

-----------------------------

-> It always create new Session object.

-> You need to explicitly flush and close session objects.

-> In single threaded environment, It is slower than getCurrentSession.

getCurrent session.

---------------------------

-> It creates a new Session if not exists , else use same session which is in current hibernate context.

-> You do not need to flush and close session objects, it will be automatically taken care by Hibernate internally.

-> In single threaded environment , It is faster than getOpenSession.

15. what is diff between session.Get() method and session.Load() method.

--------------------------------------------------------------------------------------

-> get() and load() both are session class method used for retrieving the data from the database.

-> get() and load() return the data in the object format.

get()

----------

-> if the record is not present in the database then get() always returns null.

-> get() not create proxy obj its always return actual object.

-> get() always hits the database.

-> performance wise get() is not good.

-> get() is eager initializer, get() reads or loads an obj early. so it is called early loading.

-> If you are not sure if object with id exists or not, you can use get.

load()

-------------

-> if the record is not present in the database then it returns object not found exception.

-> load() returns proxy objects and loads data only when it is actually required . so load() is better becoz its support lazy loading.

-> load() doesnt hits the database every time.

-> performance wise it is better to get().

-> load() lazy initializer.load() reads or loads an object lazely.

-> If you are sure about existence of object, you can use load.

16. what is session.save() and session.persist().

-----------------------------------------------------------

-> save() and persist() both method which we are mainly used for saves the object in the database.

save()

-------------

-> save() method will save an obj to the database and returns the id of the saved objects(primary key) in the form of serializable type.

-> save() can be used inside or outside the transaction boundries. save() is not fit for long time running transaction.

-> its take more time to execute.

-> save() method return type is serializable.

persist()

------------------

-> save/persist both the methods we can use to save/store an obj to database.

-> persist() will save an obj into database but it doesnt return id of saved obj. it doesnt return anything. its return type is void.

-> persist() can be used only within the boundary of transaction. persist() is suitable for long time running transaction. It saves data when flush is called.

-> persist() takes less time to execute.

-> persist() method return type is void.

17. what is session.upadate().

----------------------------------------

-> update() just update the record & throws hibernate exception. if record is not avialable to update.

-> this method doesnt return anything becoz its return type is void.

-> this method cannot update primary key value.

18. what is session.save or saveorupadate().

---------------------------------------------

-> save() perform only insert but saveorupdate() update the record if record is avialable otherwise insert the record in the database table.

-> this method doesnot return anything becoz its return type is void.

-> this method we can update primary key column value.

save -> Save stores object in database. It generates id for the object and returns it. If object already exists in database, it will throw an error.

saveorUpdate -> SaveOrUpdate method save the object if id does not exist. If it exists , it calls update method.

Merge()

----------------

-> merge() update the record if record is avialable otherwise insert the record in database table. it is used to add a specified obj to the first level cache.

-> through this method we can update primary key column value also.

Hibernate jpa annotation

---------------------------------

1> @Entity

-------------------

-> it is used to mark the class as persistent java class.

2> @Table

-------------------

-> it is used to provide the details of the table.

3> @Id

-----------

-> it is used to define the primary key.

4> @GeneratedValue

-------------------------

-> it is used to define the primary key generation strategy.

5> @Column

-----------------

-> it is used to define the properties of the column that will be mapped to the annotated field. you can define several properties like name, length, nullable etc.

6> @OneToOne

---------------------

-> @OneToOne annotation is used to create one to one relationship between Country and Capital entities.

7> @joinColumn

------------------------

-> it is used to specify a mapped column for joining an entity association.

8> @Inheritance

------------------------

-> For implementing inheritance in hiberante, @Inheritance annotation is used.It defines inheritance strategy to be implement for entity class hierarchy.

-> For one table per class hierarhcy,we have used Single\_Table as inheritance strategy.This annotation is defined at root level or sub hierarchy level

where different strategy is to be applied.

9> @DescriminatorColumn

--------------------------

-> This annotation is used to define discriminator column for Single\_Table and joined strategy.It is used to distinguish between different class instances.

-> This annotation is defined at root level or sub hierarchy level where different strategy is to be applied.

-> If @DiscriminatorColumn annotation is not specified,then hibernate will create a column named as “DType” and DiscriminatorType will be string.

10> @DescriminatorValue

------------------------------

-> This annotation defines value in discriminator column for that class.This can only be applied on entity concrete class.

why relationships in database.

--------------------------------------

-> we can store our application data in a database table to make it as a persistent data.

-> for eg. consider we have an application which contains doctors and patients data and we can store that data in single table. then data redundancy problem arises.

-> to reduce the data redundancy, we need to store in a two different tables.

-> in order to get the relation between the data, we also need to take the column of one table as a column in another table.

-> in hibernate application, if we create a single pojo class and if we set the data to that obj, then there is a chance of getting data redundancy.

-> data redundancy means, some data in multiple objs of a pojo class can be duplicated.

-> in order to avoid (or reduce), we divide properties of one class into(multiple) two classes and then we apply a relationship between objs of the two classes.

-> in hibernate, we can apply four types of relationships between pojo classes.

Association relationship

-------------------------------

-> in hibernate to put relations between entities i.e one obj with multiple objs we use association relationship.

-> if we want to implement association relationship there must be foriegn key relationship between child to parent.

the main advantage of association relationship is

----------------------------------------------------------

-> to perform operations on one object we can transfer that operation on another obj.

-> e.g if we want to retrieve the data from more than one table complusory we put assosiation in hibernate.

-> to create assosiation relationship with object to object complusory their a must be foreign key relationship in the database table.

there are 4 types of Association in hibernate.

----------------------------------------------------

one-to-many relationship

------------------------------

-> in this relationship one parent has multiple child object.

-> one user has multiple task.

-> country and state. One Country can have n number of states. and team and player one team has multiple players. customer and items.

-> to create this kind of association relationships in hibernate we need to perform two modification.

1.> in parent pojo class we must take child class as a collection properties. eg set properties, list properties, map properties.

2.> in parent class mapping file we must take one coressponding collection properties to map this child properties.

many-to-one relationship

---------------------------------

-> multiple tasks for one user.

many-to-many relationship

----------------------------------

-> Lets take example of Country and Language. One Country can have n number of languages and one language can be spoken by n number of countries.

Inheritance Mapping (or) inheritance Strategies

---------------------------------------------------------

-> through inheritance we can save both parent object and child obj in to the db.

-> in a hibernate application, if there are multiple POJO classes and if they have common properties, then to get reusability we apply inheritance.

-> common properties, we separate and we create in super class and we extend that super class to multiple sub classes.

CreditCard.java

----------------------

public class CreditCard {

private int paymentID;

private double amount;

private Date paymentDate;

private int cardNumber;

private String cardType;

getters & setters

}

Cheque.java

-----------------

public class Cheque {

private int paymentID;

private Date paymentDate;

private int chequeNumber;

private String chequeType;

getters & setters

}

-> in the above two POJO classes, there are three common properties. so in order to get Resuability, we apply inheritance by separating common properties

to a super class (payment) like this following.

Payment.java

------------------

public class Payment {

private int paymentID;

private double amount;

private Date paymentDate;

getters & setters

}

CreditCard.java

------------------

public class CreditCard extends Payment {

private int cardNumber;

private String cardType;

getters & setters

}

Cheque.java

---------------

public class Cheque extends Payment {

private int chequeNumber;

private String chequeType;

getters & setters

}

-> hibernate has provided three inheritance strategies, to map the classes of hierarchy to database tables.

1.> table per class:

--------------------------------

-> in this hierarchy we store both parent data and child data in a single data.

-> we need to choose this strategy, if we want to map all the classes of hierrachy to a single table of database.

-> through descriminator column we can identify which employee it is.

-> descriminator column help us to is identified which child is.

payment payment (single table)

/\

credit cheque

card

2.> table per concrete class

--------------------------------------

-> if we want to map each concrete class hierarchy to a separate table of database then we need to choose this strategy.

payment credit card (separte table) cheque (separate table)

/\

credit cheque

card

3.> table per concrete class

----------------------------------

-> if we want to map each class of hierarchy to a separate table of database, then we need to choose this strategy.

payment payment(separate table) credit card(separate table) cheque(separate table)

/\

credit cheque

card

Hql queries.

--------------------

1.> FROM Clause - FROM clause is used to load all objects into memory.

Example:

String hql = "FROM Customer";

Query<Customer> query = session.createQuery(hql, Customer.class);

List<Customer> custList = query.list();

String hql = "FROM Customer";

Query<Customer> query = session.createQuery(hql, Customer.class);

List<Customer> custList = query.list();

2.> AS Clause - AS clause is an optional keyword, used to create an aliases for a classes in HQL queries.

example:

String hql = "FROM Customer AS C";

// Above statement is the same with statement: String hql = "FROM Customer C";

Query<Customer> query = session.createQuery(hql, Customer.class);

List<Customer> custList = query.list();

String hql = "FROM Customer AS C";

// Above statement is the same with statement: String hql = "FROM Customer C";

Query<Customer> query = session.createQuery(hql, Customer.class);

List<Customer> custList = query.list();

3.> SELECT Clause - SELECT clause is used to obtain few properties of returned objects.

String hql = "SELECT C.firstName, C.age FROM Customer C";

Query query = session.createQuery(hql);

List<Object[]> objectList= query.list();

String hql = "SELECT C.firstName, C.age FROM Customer C";

Query query = session.createQuery(hql);

List<Object[]> objectList= query.list();

4.> WHERE Clause - WHERE clause is used to narrow selected objects which must meet some conditions.

String hql = "FROM Customer C WHERE C.age = 20";

Query<Customer> query = session.createQuery(hql, Customer.class);

List<Customer> custList = query.list();

String hql = "FROM Customer C WHERE C.age = 20";

Query<Customer> query = session.createQuery(hql, Customer.class);

List<Customer> custList = query.list();

5.> ORDER BY Clause - ORDER BY clause is used to sort HQL query’s results.

String hql = "FROM Customer C WHERE C.age > 25 ORDER BY C.id DESC";

Query<Customer> query = session.createQuery(hql, Customer.class);

List<Customer> custList = query.list();

String hql = "FROM Customer C WHERE C.age > 25 ORDER BY C.id DESC";

Query<Customer> query = session.createQuery(hql, Customer.class);

List<Customer> custList = query.list();

6.> GROUP BY Clause - GROUP BY clause is used with aggregate functions.

String hql = "SELECT COUNT(\*), C.firstName FROM Customer C " + "GROUP BY C.firstName";

Query query = session.createQuery(hql);

List<Object[]> objLst = query.list();

String hql = "SELECT COUNT(\*), C.firstName FROM Customer C " + "GROUP BY C.firstName";

Query query = session.createQuery(hql);

List<Object[]> objLst = query.list();

7.> Named Parameters - Named Parameters is used to get inputs from users and helpful to defend against SQL injection attacks.

String hql = "FROM Customer C WHERE C.id = :customerId";

Query<Customer> query = session.createQuery(hql, Customer.class);

query.setParameter("customerId",1);

List<Customer> custList = query.list();

String hql = "FROM Customer C WHERE C.id = :customerId";

Query<Customer> query = session.createQuery(hql, Customer.class);

query.setParameter("customerId",1);

List<Customer> custList = query.list();

8.> UPDATE Clause - UPDATE clause is used to modify properties of objects.

String hql = "UPDATE Customer set age = :custAge WHERE id = :customerId";

Query query = session.createQuery(hql);

query.setParameter("custAge",26);

query.setParameter("customerId",3);

int affectedRows = query.executeUpdate();

9.> DELETE Clause - DELETE clause is used to delete objects.

String hql = "DELETE FROM Customer WHERE id = :customerId";

Query query = session.createQuery(hq0l);

query.setParameter("customerId", 2);

int affectedRows = query.executeUpdate();