**Hibernate**

1. Hibernate is one of the best ORM frameworks available for java
2. ORM means Object Relational mapping
3. The process of storing data is called as persistence
4. In ORM we store the data in the object form into the table
5. To use ORM, we have to use many tools which are known as ORM tools, the some of the tools are hibernate, iBatis, TopLink, etc.
6. To save the object in the database we use the save(obj) method, which further requires session and which further requires the session factory, and in session factory we have to provide the configuration for the database.
7. For configuring the session factory we can use many languages like XML, Java Configuation way, etc
8. We have to provide the driver name, username, password, driver name, database url in the configuration so we can load it inside the session factory object
9. For making the object to be saved directly, we have to add the following statements to the appTest or default java file:

Configuration confg=new Configuration();

confg.configure(“hibernet.cfg.xml”);

SessionFactory sf=configuration.buildSessionFactory();

Session session=sf.openSession();

Transation tr=session.beginTransation();

Session.save(<bean object name>);

tr.commit();

session.close();

1. And then we have to create an java class by the name of the objects and then make it bean class of POJO by declaring the private variables and then put getter setter method in it.
2. To provide the data regarding the dbms to hibernate, we have to make an hibernate.cfg.xml file and while making it we have to enter the right details and we have to add the following line into it

<property name=”hbm2ddl.auto”>update</property>

<mapping class=”Alien”/>

1. (IMP)Select or change the database name in hibernate.cfg.xml file in .dialet to mysql5 from mysql…
2. Instead of using the hibernate.cfg.xml use the annotations example
3. (NOT MANDATOR CAN ALSO BE DONE BY THE HELP OF ANNOTATION IN WHICH MAKE @Entity ABOVE CLASS THEN @Table(name=”<Table name>”) & FOR PRIMARY KEY @Id above the primary variable)Then we have to make an hibernate.hbm.xml file in which we have to write the following content OR we can make the new file by the name of the <Entity-Name>.hbm.xml and then add the sentence <mapping resource=”<Direction Here>”> in the main xml file and add the following code in it between <hibernate-mapping>….</hibernate-mapping>

<class name=”Alien” table=”alien”>

<id name=”aid” column=”aid”>

<generator class=”assigned”></generator>

</id>

<property name=”name” column=”name”/>

<property name=”color” column=”color”/>

</class>

1. If the error comes of the file not found then it is due to maven, make all the files under the default package and then run it, the error has to be gone
2. If the jars are not to be added even after adding the text to the pom.xml file then in eclipse go to the projects -> clean and make the selection of the project by selecting start building immediately, and tada the jars will come to the maven libray..
3. If the error comes that the column not found and the could not execute the statement then there is some fault in the table of the database in which the particular row
4. Annotations
   1. @Entity -use to mark any class as Entity
   2. @Table -use to change the table details
   3. @Id -use to mark column as id(primary key)
   4. @GeneratedValue -hibernate will automatically generate values for that using an internal sequence. Therefore we don’t have to set it manually
   5. @Column -can be used to specify column mappings. For example to change the name in the associated table in database
   6. @Transient -This tells hibernate not to save this field]
   7. @Temporal -tells the hibernate to save the given date or time in the formatted way
   8. @Lob -tells hibernate that this is the large object not the small object
   9. @OneToOne, @OneToMany, @ManyToOne, @JoinColoumn, etc
5. If we do not specify the name of the table in the hibernate then the table will be made of the name of the bean class itself
6. (I.M.P.)

|  |  |
| --- | --- |
| get() | load() |
| get method of the hibernate session returns null if the object is not found in cache as well as on database. | load() method throws ObjectNotFoundException if the object is not found on cache as well as on database but never return null. |
| get() involves databse hit if object doesn’t exists in Session cache and returns a fully initialized object which may involve several database call | load method can return proxy in place and only initialize the object or hit the database if any other than getId() is called on persistence or entity object. This lazy initialization increases the performance. |
| Use if you are not sure that object exists in the database | Use if you are sure that object exists |

1. In hibernate to get or read the object the following sentences are used:

Student st=(Student)session.load(Student.class, 2)

Address ad=(Address)session.get(Address.class, 2);

1. By the use of the @Embeddable we can continue to make column of the variable of the other class

Eg. @Entity

Class Student @Embeddable

{ class Certificate

@Id {

Private Int sid; String course;

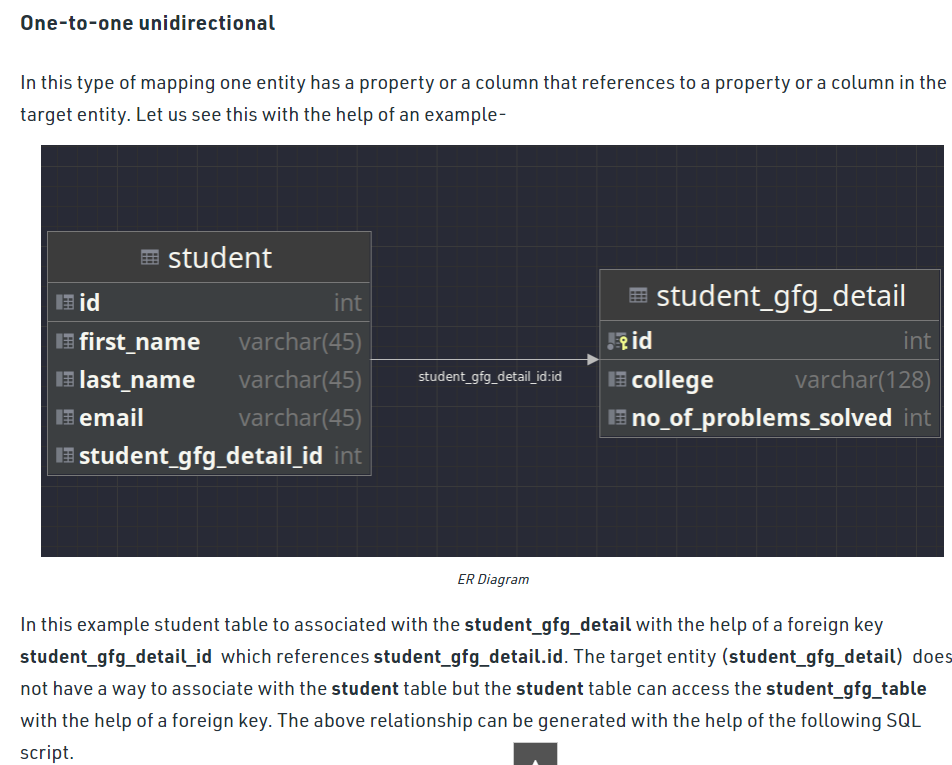
Private String name; String duration;

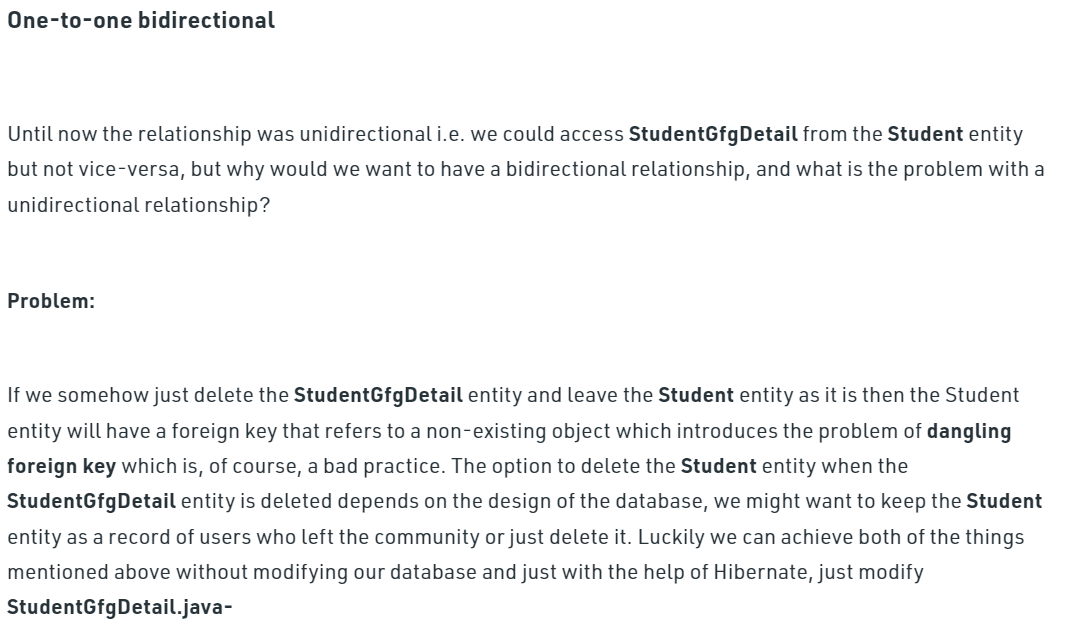
Private Certificate certi; }

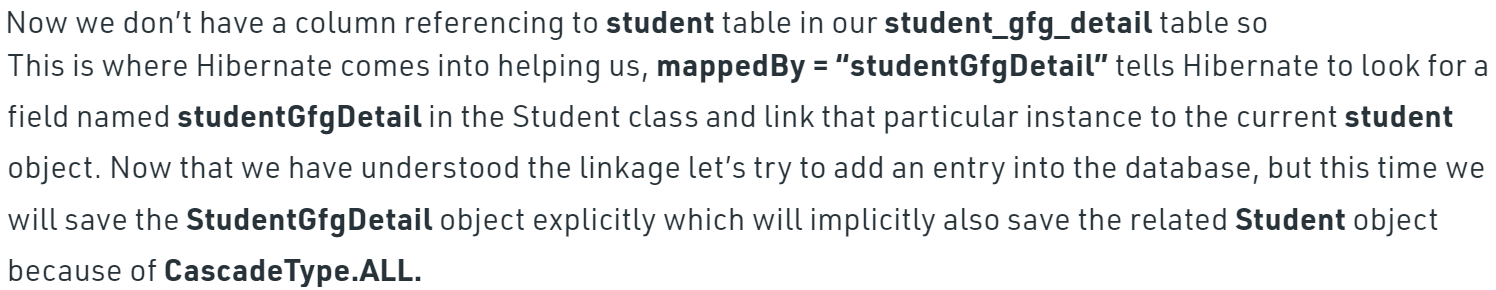
}

Hence, in the table Student the colums will be sid, name, certi, course, duration

1. Hibernate mappings are one of the key features of the hibernate, They establish the relationship between two databases tables as attributes in your model
2. Hibernate allows you to easily navigate the associations in your model and criteria quries
3. The mapping are of the two types the uni- directional and bi-directoinal

🡺





1. The mapping is determined by the annotations like @OneToOne, @OneTo many, etc
2. In the mapping the column of the one is included in the table of the other
3. By using the mappedBy=”<name>” one can make sure the new table of the following private variable is not maked

@OneToOne(mappedBy=”answer”)

Private Question question;

\*Here the annotations are used as per the requirement

1. In which we have to do many to many mapping we have to make an list in the opp class of the many one
2. By using @JoinTable(name=””) one can change the name of the new table to be formed
3. One can also change the name of the column of the newly forming table by joining the two tables by the following code :

@ManyToMany

@JoinTable(name=”emp\_learn”

joinColumns = {@JoinColumns(name=”eid”)},

inverseJoinColumns = {@JoinColumn(name=”pid”)}

)

Private List<Project> projects;

1. For fetching the data there are two types
   1. Lazy Type : In lazy loading, associated data loads only when we explicitly call getter or size method.
   2. Eager type : It is a design patten in which data loading occurs on the spot.
2. For fetching data from the database via the hibernate we have to first store the data first in the object of the class and then by using the get method one can use it.
3. The database we use is called as RDMS(Relational Database Management System)
4. In hibernate there are four states in life cycle
   1. Transient State
   2. Persistent State
   3. Detached State
   4. Removed State
5. When the hibernate is in the persistence state then at that time after saving the object in the database by the help of the session.save(), if we chage the value of the one of the variable of the object by the help of the set method then the value of it will be automaticly saved in the database without doing any thing. To stop it we have to stop session.
6. If we want to delete the data from the database then we have to first use the .get() method and then use the session.delete() method to remove the data from the database, but the data will be remained in the session till it is not closed
7. Difference between HQL and SQL

|  |  |
| --- | --- |
| **HQL(Hibernate Query Language)** | **SQL(Sequal Query Language)** |
| It is database independent | It is database dependent |
| Easy to learn for programmer | Easy to learn for DBA |
| From Student  (Where Student is the name of entity i.e. class, not table name) | Select \* from Student  (Where Student is the name of the table) |

1. (I.M.P.) The important point is that in HQL we have to write the name of the entity instead of the name of the class
2. The simple syntax used for HQL are :

String query=”from Sudent”;

Query q=session.createQuery(query);

List<Student> list=q.list(); (For the multiple-list)

For(Student stud : list)

{

System.out.println(stud.getName());

}

1. The results can be of the two types:
   1. Single (Unique)
   2. Multiple (List)

String query=“from Student where city=:c and name=: n”;

Query q=session.createQuery(query);

q.setParameter(“c”,”ABC”);

q.setParameter(“n”,”peter”);

List<Student> list=q.list();

For(for\_each loop){…}

{

1. For the complex queries in the HQL

String query=”from Student where city=’lucknow’”

Query q=session.createQuery(query);

List<Student> list=q.list();

For(Student stud : list)

{

System.out.println(stud.getName());

}

1. For making the transaction to occur one should use transaction.commit(); at the end but if one want to roll back the all the sentence that have occurred before if any error is occues than the transation.rollBack(); should be done to cancel all the further quries that have been taken place
2. For making the update the following statements can be used :

Transaction tr=session.beginTransaction();

Query q2=session.createQuery(“update Student set city=:c where name:=n”);

q2.setParameter(“c”,”Pune”);

q2.setParameter(“n”,”sundar”);

int i=q2.executeUpdate();

tr.commit();

1. And in the query of the HQL we have to use name of the variables used in the entity instead of the name of the column of the table
2. For making the result of the reqired purpose using the HQL one should use the code like following :

Query q3=session.createQuery(“select q.question, q.questionId, a.answer from Question as q INNER JOIN q.answers as a”);

List<Object []> list3=q3.getResultList();

1. (I.M.P.) While using the HQL in the hibernate we would undergoe throught the exception that query in the org.hibernate.\* package is been deprecated, The new interface Query is been introduced in the package org.hibernate.Query.\*; So change the package which is been used by default
2. **To divide a large number of records into multiple parts**, we use pagination. It allows users to display a part of records only. Loading all records on a single page may take time, so it is always recommended to created pagination.
3. We can control the amount of the results the HQL is showing use by the help of the following codes:

Query.setFirstResult(0);

🡪Where the 0 is the index number in the table from where we want to start to fetch the data

Query.setMaxResult(10)

🡪Here the 10 is the max number of the result we want to fetch from the database.

1. In hibernate one can use the SQL quries also by the following type of codes :

String sql=”select \* from Student”;

NativeQuery nq=session.createSQLQuery(sql);

List<object[]> list=nq.list();

For(Object[] st : list)

{

System.out.println(st[2]);

}

1. Cascade is used when we have to perform the same operation to the connecting variable/object that we have used with the entity, We donot have to make that function differently for it

Eg. (In the entity class of Question)

@OneToMany(mappedBy=”question” cascade=CascadeType.All)

Private List<Answer> answers;

\*Here instead of the all, we can use many of the functions….

1. There are two level of cache in the hibernate
   1. Primary Cache(The default one)
   2. Secondary Cache(One have to configure it)
2. For implementing the Second level cache,
   1. we first have to add the two jar files additional to the hibernate orm jar file to the project which are ehcache and hibernate cache
   2. Then we have to add the following statement to the hibernate.cfg.xml file :

<property name=”cache.use\_second\_level\_cache”>true</property>

<property name=”cache.region.factory\_class”>org.hibernate.cache.ehcache.internal.EhcacheRegionFactory</property>

(\*The above package name of the second property can be also taken by writing the EhCacheRegionFactory by the ehcache.hibernate class and importing the the required class, we would get the required name and we will use it and remove it)

* 1. By default the entity we are using is not cacheable and hence first we have to add the annotation @Cacheable, @Cache(usage= CacheConcurrencyStratergy.READ\_ONLY)

(\* Instead of the READ\_ONLY we can use many of the methods)

51. By the help of the Criteria one can fetch the data without the use of the query and the example of the criteria is as follow :

Criteria c=session.createCriteria(Student.class);

c.add(Restrictions.eq(“certi.course”,”Android”));

List<Student> stud=c.list();

For(Student s : stud)

{

System.out.println(st);

}

(\* Here instead of the .eq we can use many of the methods like .gt, .lt, .like, .ilike)

52. Projection\*