-----Hibernate----

Low level JDBC code is eliminated

Added hibernate jar files from course and mysql jdbc connector jar from open source

Entity class - Java class mapped to a data base table ( like a value object POJO class ) but adding special annotations to them

To map Normal Java class to data base table we have following options:

1. XML config ( Legacy )

2. Java annotations ( new, prefered )

**MySQL Connection created for this work:**

*Uname*: hbstudent

*Pwd*: hbstudent

Java annotations method steps:

step1: map java class to database table

step2: map fields to database columns

Session factory:

a. reads the hibernate config file

b. creates session objects

c. heavy weight object

d. only created once in the app

Session:

a. wraps a JDBC connection

b. main object used to save/retrieve objects

c. short-lived object

d. retrieved from session-factory

Primary keys in hibernate: We can generate different types of auto generated values with diff options available in hibernate

**CRUD Operations**

Main operation for hibernate for all CRUD operations:

- opening session factory for the class (SessionFactory sessionFactory = new Configuration()

.configure("hibernate.cfg.xml")

.addAnnotatedClass(Student.class)

.buildSessionFactory())

- getting current sessions for each session/transaction (sessionFactory.getCurrentSession())

- begin the transaction with the help of session (session.beginTransaction())

- Create/read/update/delete

- Commit for each transaction (session.getTransaction().commit())

HQL is similar to SQL

Create: creating by entering the values except primary key session.save() is used for creating row

Read: Read single record/row by the primary key Using session.get() ,

session.createQuery() is used to retrieve multiple rows/objects

Update: object.Setter is used for single row update ,

session.createQuery().executeUpdate is used to update bulk records

Delete: Can be done by either session.createQuery().executeUpdate by writing delete query or else directly session.delete("")

For setting up log4j in the project : https://www.udemy.com/course/spring-hibernate-tutorial/learn/lecture/5835894#content

Every time, when we copy the existing project and pasting it making it a new project, we need to make the following changes for changing the schema in:

1. Hibernate config file
2. TestJDBC.java file where we define database settings

**One-to-One mappings**

Uni directional & Bi-directional

Cascade:

Changing a data from one table that will affect data in other table.

cascade types: It is defined in the entity class as @OneToOne (cascade=Cascadetype.ALL)

* PERSIST, REMOVE, REFRESH, DETACH, MERGE, ALL ( Combi of all the other cascade types )

By default, no operations are cascaded

Reverse Engineer in mysql : Database -> reverse engineer … : This will give us the diagrams for the tables we created

***@JoinColumn*** : Is used for mapping the current table column to the primary key of other table. ( Primary key of other table that is mapped )

- If the join is for a OneToOne or ManyToOne mapping using a foreign key mapping strategy, the foreign key column is in the table of the source entity or embeddable.

- If the join is for a unidirectional OneToMany mapping using a foreign key mapping strategy, the foreign key is in the table of the target entity.

- If the join is for a ManyToMany mapping or for a OneToOne or bidirectional ManyToOne/OneToMany mapping using a join table, the foreign key is in a join table.

- If the join is for an element collection, the foreign key is in a collection table.

***@GeneratedValue()***: The GeneratedValue annotation may be applied to a primary key property or field of an entity or mapped superclass in conjunction with the [Id](https://docs.jboss.org/hibernate/jpa/2.1/api/javax/persistence/Id.html) annotation. The use of the GeneratedValue annotation is only required to be supported for simple primary keys. Use of the GeneratedValue annotation is not supported for derived primary keys ( Primary key of current table )

*Bi-Directional method*: Changing data in dependent table should change the data in the parent table. We use **@OneToOne(mappedBy=””)** for bi-directional mapping in the entity classes

Session.delete() will delete both the table entries if there is still cascade.ALL

To delete only one table entry change the CASCADEtype from ALL to other types except REMOVE. And add set the data object ( which we still need to persisit ) as NULL in main app ( see code DeleteInstructorDemo.java)

**One-to-Many mappings:**

Having one instructor for many courses is one to many mapping ( or vice versa of one course taught by many instructors ) If we delete one instructor we should not be deleting the course.

@OneToMany(mappedBy=”instructor”) : Used to map one to many relationship in entity class and we don’t use Cascadetype.REMOVE mentioned so that the associated instructor

MappedBy tells hibernate:

* Look at the instructor property in Course class
* Use information from Course class @JoinColumn
* To help find associated courses for instructor

**Fetch types:** Type of pulling the data (FetchType.EAGER/LAZY : in the entity class where @Onetoone/many mappings are noted.

Eager:

Which will retrieve everything, dependent entities

Lazy:

Will retrieve on request. Will load the main entity first and the dependent entities on demand (Lazy)

To get the lazy loading data – we need to have an open hibernate session

**Many to Many mappings:**

**It used** @JoinTablewhich has the info for the table that is used for joining the many to many relationships of the tables. We use @joincolumn & @inversejoincolumn ( Other table where it is supposed to be joined along with the current table )

@jointable eliminates using of @mappedby

In @manytomany mappings, deletion of one table data won’t affect other table data as we are using @jointable