Table of Contents

Java Code :

```
Student student = new Student(1,"Mijan");

SessionFactory factory = new Configuration().configure().buildSe
ssionFactory();
Session session = factory.openSession();
session.beginTransaction();
session.save(student);
session.getTransaction().commit();
session.close();
```

```
@Entity (name="Student_Class")
public class Student{
    @Id
    @Column(name="Student_Id")
    int id;
    @Column(name="Student_Name")
    String name;

    pubic Student(){
    }

    @Id
    @Column(name="Student_Id")
    public int getId(){
        return id;
    }

    public void setId(int id){
        this.id = id;
    }

    @Column(name="Student_Name")
    public String getName(){
        return name;
    }

    public void setName(){
        this.name = name;
    }
}
```

```
@Entity
@Table (name = "Student_Table")
public class Student {
    @Id @GeneratedValue (strategy = GenerationType.AUTO)
    int id;
    String name;
    @Temporal(TemporalType.DATE)
    Date date;
    @Lob
    String Description;
    @Transient
    String email;
```

```
Student student = new Student(1,"Mijan");

SessionFactory factory = new Configuration().configure().buildSe
ssionFactory();
Session session = factory.openSession();
session.beginTransaction();
session.save(student);
session.getTransaction().commit();
session.close();

student = null;
session = factory.openSession();
student = (Student) session.get(Student.class, 1);
System.out.println(student.toString());
```

```
Student student = new Student(1,"Mijan");
Address address = new Address("Dhaka", "Rangpur");
student.setAddress(address);

SessionFactory factory = new Configuration().configure().buildSe
ssionFactory();
Session session = factory.openSession();
session.beginTransaction();
session.save(student);
session.getTransaction().commit();
session.close();
```

```
public class Student{
    @Id
    int id;
    String name;
    @Embedded
    Address address;
}
```

```
@Embeddable
public class Address {
    String presentAddress;
    String permanentAddress;
}
```

There will be one and only table in the database.

```
@Entity
public class Student {
    @Id
    int id;
    String name;
    @Embedded
    Address address;
    @Embedded
    @AttributeOverrides ({
        @AttributeOverride (name = "presentAddress", column = @C
olumn(name = "homePresentAddress")),
        @AttributeOverride (name = "permanentAddress", column =
@Column(name = "homePermanentAddress"))
    })
    Address homeAddress;
}
```

There will be one and only table in the database.

```
Student student = new Student(1,"Mijan");
Address address = new Address("Dhaka", "Rangpur");
Address address1 = new Address("Bangladesh", "Bangladesh");
student.getMyList().add(address);
student.getMyList().add(address1);

SessionFactory factory = new Configuration().configure().buildSe
ssionFactory();
Session session = factory.openSession();
session.beginTransaction();
session.save(student);
session.getTransaction().commit();
session.close();
```

```
@Entity
public class Student {
    @Id
    int id;
    String name;
    @ElementCollection
    List<Address> myList = new ArrayList<>();
}
```

```
@Embeddable
public class Address {
    String presentAddress;
    String permanentAddress;
}
```

Saving Collection will make an Address table Student Id will be there foreign key.

```
@Entity
public class Student {
    @Id
    int id;
    String name;
    @ElementCollection
    @JoinTable (name = "Address_List", joinColumns = @JoinColumn
 (name = "StudentId"))
    @GenericGenerator(name="hilo-gen", strategy = "hilo")
    @CollectionId (columns = {@Column (name = "CollectionList")}
, generator = "hilo-gen", type = @Type (type = "long"))
    List<Address> myList = new ArrayList<>();
}
```

This will create three table one for Student table one for collection of Address and another is generator unique key.

```
Student student = new Student(2,"Mijan");
Address address = new Address("Dhaka", "Rangpur");
Address address1 = new Address("Bangladesh", "Bangladesh");
student.getMyList().add(address);
student.getMyList().add(address1);

SessionFactory factory = new Configuration().configure().buildSe
ssionFactory();
Session session = factory.openSession();
session.beginTransaction();
session.save(student);
session.getTransaction().commit();
session.close();

student = null;
session = factory.openSession();
student = (Student) session.get(Student.class, 2);
session.close();
System.out.println(student.getMyList().size());
```

```
@Entity
public class Student {
    @Id
    int id;
    String name;
    @ElementCollection (fetch = FetchType.EAGER)
    @JoinTable (name = "Address_List", joinColumns = @JoinColumn
 (name = "StudentId"))
    List<Address> myList = new ArrayList<>();
}
```

Lazy Method : Lazy method normally doesn't pull the full data associated with the object . Normally it runs a senod query if you need whole data associated with the object.

Eager Method : Eage method normally pull up full data associated with the object.

If you want to pull all the data then you will use Eager method otherwise you will use Lazy method.

```
public Hibernate() {
        Student student = new Student(2,"Mijan");
        Address address = new Address("Dhaka", "Rangpur");
        student.setAddress(address);

        SessionFactory factory = new Configuration().configure()
.buildSessionFactory();
        Session session = factory.openSession();
        session.beginTransaction();
        session.save(student);
        session.save(address);
        session.getTransaction().commit();
        session.close();

    }
```

```
@Entity
public class Student {
    @Id
    int id;
    String name;
    @OneToOne
    Address address;
}
```

```
@Entity (name = "Address")
public class Address {
    @Id
    String presentAddress;
    String permanentAddress;
}
```

```
 public Hibernate() {
        Student student = new Student(2,"Mijan");
        Address address = new Address("Dhaka", "Rangpur");
        student.getMyAddress().add(address);
        address.setStudent(student);

        SessionFactory factory = new Configuration().configure()
.buildSessionFactory();
        Session session = factory.openSession();
        session.beginTransaction();
        session.save(student);
        session.save(address);
        session.getTransaction().commit();
        session.close();

    }
```

```
@Entity
public class Student {
    @Id
    int id;
    String name;
    @OneToMany (mappedBy = "student")
    @JoinTable (name = "many", joinColumns = @JoinColumn (name =
"StudentId"), inverseJoinColumns = @JoinColumn (name = "AddressI
d"))
    List<Address> myAddress = new ArrayList<>();
}
```

```
@Entity (name = "Address")
public class Address {
    @Id
    String presentAddress;
    String permanentAddress;
    @ManyToOne
    @JoinColumn (name = "StudentId")
    @NotFound(action = NotFoundAction.EXCEPTION)
    Student student;
}
```

One To Many relationship in Hibernate make two separate table for two entity and also create another table in which foreign key of each entity is saved.

Not Found annotation works when you don't have any Student Object for Address Object to avoid this exception we use Not Found annotation.

```
    public Hibernate() {
        Student student = new Student(3,"Mijan");
        Address address = new Address("Dh1aka", "Rangpur");
        student.getMyAddress().add(address);
        address.getListOfStudent().add(student);
        student.getMyAddress().add(address);

        SessionFactory factory = new Configuration().configure()
.buildSessionFactory();
        Session session = factory.openSession();
        session.beginTransaction();
        session.save(student);
        session.save(address);
        session.getTransaction().commit();
        session.close();

    }
```

```
@Entity
public class Student {
    @Id
    int id;
    String name;
    @ManyToMany (mappedBy = "listOfStudent")
    List<Address> myAddress = new ArrayList<>();
}
```

```
@Entity (name = "Address")
public class Address {
    @Id
    String presentAddress;
    String permanentAddress;
    @ManyToMany
    List<Student> listOfStudent = new ArrayList<>();
}
```

```
 public Hibernate() {
        Student student = new Student(3,"Mijan");
        Address address = new Address("Dh1aka", "Rangpur");
        student.getMyAddress().add(address);

        SessionFactory factory = new Configuration().configure()
.buildSessionFactory();
        Session session = factory.openSession();
        session.beginTransaction();
        session.persist(student);
        session.getTransaction().commit();
        session.close();

    }
```

```
@Entity
public class Student {
    @Id
    int id;
    String name;
    @OneToMany(cascade = CascadeType.PERSIST)
    List<Address> myAddress = new ArrayList<>();
}
```

```
@Entity (name = "Address")
public class Address {
    @Id
    String presentAddress;
    String permanentAddress;
}
```

If you use cascade type then you don't need to save all object associated with object you want to save.Cascade will take care of that.

```
   public Hibernate() {
        Student student = new Student(3,"Mijan");
        GoodStudent goodStudent = new GoodStudent("Good Student",
1,"Rifat");
        BadStudent badStudent = new BadStudent("BadStudent",2,"M
ijan");


        SessionFactory factory = new Configuration().configure()
.buildSessionFactory();
        Session session = factory.openSession();
        session.beginTransaction();
        session.save(student);
        session.save(goodStudent);
        session.save(badStudent);
        session.getTransaction().commit();
        session.close();

    }
```

```
@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name = "Student_Type", discriminatorType =
DiscriminatorType.STRING)
public class Student {
    @Id
    int id;
    String name;
}
```

```
@Entity
@DiscriminatorValue (value = "Good")
public class GoodStudent extends Student{
    String quality;
}
```

```
@Entity
@DiscriminatorValue (value = "Good")
public class GoodStudent extends Student{
    String quality;
```

```
@Entity
@DiscriminatorValue (value = "Bad")
public class BadStudent extends Student{
    String quality;
}
```

Inheritance Single Table map all Classes into a single table.

Discriminator Value help us to configure DTYPE Column and the value.

```
public Hibernate() {
   Student student = new Student(3,"Mijan");
   GoodStudent goodStudent = new GoodStudent("Good Student",1,"R
ifat");
   BadStudent badStudent = new BadStudent("BadStudent",2,"Mijan"
);


   SessionFactory factory = new Configuration().configure().buil
dSessionFactory();
   Session session = factory.openSession();
   session.beginTransaction();
   session.save(student);
   session.save(goodStudent);
   session.save(badStudent);
   session.getTransaction().commit();
   session.close();

}
```

```
@Entity
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
@Inheritance(strategy = InheritanceType.JOINED)
public class Student {
    @Id
    int id;
    String name;
}
```

```
@Entity
public class GoodStudent extends Student{
    String quality;
}
```

```
@Entity
public class BadStudent extends Student{
    String quality;
}
```

Table Per Class strategy simply crate table for each Entity.

Joined Strategy create a table where every class got an id and there is another class for each entity.

Create:

```
public Hibernate() {

    SessionFactory factory = new Configuration().configure().bui
ldSessionFactory();
    Session session = factory.openSession();
    session.beginTransaction();
    for(int i=1; i<=10; i++){
        Student student = new Student();
        student.setName("User name is: " + i);
        session.save(student);
    }
    session.getTransaction().commit();
    session.close();

}
```

Retrive:

```
public Hibernate() {

    SessionFactory factory = new Configuration().configure().bui
ldSessionFactory();
    Session session = factory.openSession();
    session.beginTransaction();
    Student student = (Student) session.get(Student.class, 3);
    System.out.println(student.getName());
    session.getTransaction().commit();
    session.close();

}
```

Update:

```
public Hibernate() {

    SessionFactory factory = new Configuration().configure().bui
ldSessionFactory();
    Session session = factory.openSession();
    session.beginTransaction();
    Student student = (Student) session.get(Student.class, 3);
    session.Update(student);
    session.getTransaction().commit();
    session.close();

}
```

Delete:

```
public Hibernate() {

    SessionFactory factory = new Configuration().configure().bui
ldSessionFactory();
    Session session = factory.openSession();
    session.beginTransaction();
    Student student = (Student) session.get(Student.class, 3);
    session.delete(student);
    session.getTransaction().commit();
    session.close();

}
```

Transient Object : Befor Saving an object using session.save we call that object as a Transient object.

Persist Object : Once we save an object using session.save we call that object is Persisit object.

Deatched Object : After closing the session we call that object as a deatched object.

```
public Hibernate() {

    SessionFactory factory = new Configuration().configure().bui
ldSessionFactory();
    Session session = factory.openSession();
    session.beginTransaction();
    Student student = (Student) session.get(Student.class, 3);
    session.getTransaction().commit();
    session.close();

    student.setName("User name Updated");

    session = factory.openSession();
    session.beginTransaction();
    session.update(student);
    session.getTransaction().commit();
    session.close();

}
```

```
@Entity
@org.hibernate.annotations.Entity (selectBeforeUpdate = true)
public class Student {
    @Id @GeneratedValue(strategy = GenerationType.AUTO)
    int id;
    String name;
 }
```

Once an object is deatched we can make that object to persisit. if you run a update query but we didn't change anything to the object then we nee org.hibernte.annotations.Entity to check whether run update query or not.

```
public Hibernate() {
    SessionFactory factory = new Configuration().configure().bui
ldSessionFactory();
    Session session = factory.openSession();
    session.beginTransaction();
    Query query = session.createQuery("from Student where id > 5"
);
    List student = query.list();
    session.getTransaction().commit();
    session.close();
    System.out.println("The size of the list is: " + student.siz
e());

}
```

```
public Hibernate() {

    SessionFactory factory = new Configuration().configure().bui
ldSessionFactory();
    Session session = factory.openSession();
    session.beginTransaction();
    Query query = session.createQuery("select name from Student "
);
    //Query query = session.createQuery("select max(id) from Stu
dent");
    //Query query = session.createQuery("select map(id,name) fro
m Student");
    query.setFirstResult(5);
    query.setMaxResults(5);
    List<Student> list = query.list();
    session.getTransaction().commit();
    session.close();
    for(Student s : list)
        System.out.println(s);

}
```

```
public Hibernate() {

    SessionFactory factory = new Configuration().configure().bui
ldSessionFactory();
    Session session = factory.openSession();
    session.beginTransaction();
    String id = "5";
    String username = "User name is: 3";
    Query query = session.createQuery("from Student where id <:u
serId and name = :userName");
    query.setInteger("userId", Integer.parseInt(id));
    query.setString("userName", username);

    List<Student> list = query.list();
    session.getTransaction().commit();
    session.close();
    for(Student s : list)
        System.out.println(s);

}
```

```
public Hibernate() {

    SessionFactory factory = new Configuration().configure().bui
ldSessionFactory();
    Session session = factory.openSession();
    session.beginTransaction();
    String id = "5";
    String username = "User name is: 3";
    Query query = session.getNamedQuery("Student.byName");
    query.setString("id", username);

    List<Student> list = query.list();
    session.getTransaction().commit();
    session.close();
    for(Student s : list)
        System.out.println(s);

}
```

```
@Entity
@NamedQuery(name = "Student.byId", query = "from Student where i
d = :id")
@NamedNativeQuery(name = "Student.byName", query = "select * fro
m Student where name = :id", resultClass = Student.class)
public class Student  {
    @Id @GeneratedValue(strategy = GenerationType.AUTO)
    int id;
    String name;
}
```

Named query allow HQL Query and Named Native Query allow SQL Query.

```
public Hibernate() {

    SessionFactory factory = new Configuration().configure().bui
ldSessionFactory();
    Session session = factory.openSession();
    session.beginTransaction();
    Criteria criteria = session.createCriteria(Student.class);
    criteria.add(Restrictions.or(Restrictions.between("id", 0, 3
), Restrictions.between("id", 7, 10)));
    criteria.add(Restrictions.eq("name", "User name is: 3")).add
(Restrictions.like("name", "%3"));


    List<Student> list = criteria.list();
    session.getTransaction().commit();
    session.close();
    for(Student s : list)
        System.out.println(s);

}
```

We can add restrictions after one restriction we can also add all Normal SQL Restrictions here.

```
public Hibernate() {

    SessionFactory factory = new Configuration().configure().bui
ldSessionFactory();
    Session session = factory.openSession();
    session.beginTransaction();
    Student student = new Student();
    student.setId(1);
    student.setName("mijan");
    Example example = Example.create(student).excludeProperty("n
ame");
    Criteria criteria = session.createCriteria(Student.class).ad
d(example);
        //addOrder(Order.desc("id"));
            //.setProjection(Projections.property("id"))
                        //.setProjection(Projections.avg("id")).
setProjection(Projections.max("id"));



    List<Student> list = criteria.list();
    session.getTransaction().commit();
    session.close();
    for(Student s : list)
        System.out.println(s);

}
```

We can add Projection and different other fuction of projection and also Example query.

Example query doesn't work with null key and primary key.

Suppose we have two session and if we run a select query then they will execute two select query.But if we use Second Level of Cache then second select query will not run except of that the object will be in the cache memory and second sessiion will work with that without using and second select statement.