



Hibernate ORM Tool

Smart Data Persistence using ORM

MKJ IT Solutions

What we cover

- In this course we will understand the ORM requirement , the need of ORM and how to prepare our machine for development.
- We will also cover the few basic annotations and mappings

What Audience will achieve after this course

- After the completion of this section the participant will be able to implement the ORM specific guidelines to implement data persistency.
- Participant will be able to understand and implement the configuration required for the development.
-

Objectives to be cover

- Understand the ORM and its need.
- Machine setup
- Basic Introduction.
- JPA Annotations and the way to persist data
- Mapping composite and collection types.
- Entity association

What is Hibernate?

- Many attempts have been made to bridge relational and object-oriented technologies or to replace one with the other, but the gap between the two is one of the hard facts of enterprise computing today. It is this challenge—to provide a bridge between relational data and Java objects—that Hibernate takes on through its object/relational mapping (ORM) approach. Hibernate meets this challenge in a very pragmatic, direct, and realistic way.

Introduction

- Hibernate is flexible and configurable ORM Solution.
- Hibernate is an open source solution for Java persistence.
- Its is based on the idea of ORM, which is a technique to map object model with relational model.
- So through hibernate we can map associated,inherited , polymorphic, composite or collection based objects directly to relational database.
- Working with objects and relational database could be complex, hibernate makes it easier.

How Hibernate helps us

- Database independent.
- Different types of Mapping
- HQL
- State management
- Native SQL Query and Criteria Query
- Cache
- Various Primary key strategies.
- Helps over JDBC issues

Setting up Development environment

- What we need.
 - Eclipse IDE (or any java specific IDE).
 - Hibernate Plugin support. (Jboss tools)
 - Hibernate Jars or Maven support.
 - Database (pfb the download link of Oracle XE)
 - <https://www.oracle.com/technetwork/database/database-technologies/express-edition/downloads/xe-prior-releases-5172097.html>
 - Understanding of Java and JDBC.

Create Basic Application

- Create cfg file
- Pojo
 - XML approach
 - @Annotation approach
- Database execution code

Understanding Session Factory

- <https://docs.jboss.org/hibernate/orm/3.5/api/org/hibernate/SessionFactory.html>
- SessionFactory is immutable instance.
- Usually a single instance per application.
- Instances of SessionFactory are thread-safe and typically shared throughout an application.
- Use to retrieve session.
- Used to contain connection information such as hibernate configuration information, mapping files, location path.

How to access Session Factory

```
Configuration cfg = new Configuration();  
SessionFactory factory = cfg.configure().buildSessionFactory();  
Session session = factory.openSession();  
Transaction t = session.beginTransaction();
```

- Hibernate sessionFactory.openSession() method used to create new session.
- We can close this session once we done with database operations.
- In best practice : we should create one session for each request in multi-threaded application or web application.

Understanding Annotations

- @ Entity
 - @ Table
 - @ ID
 - @ GeneratedValue and its Strategies
 - @ Column
 - @ OrderBy
- This annotations are from
javax.persistence.*

Usage of Annotations

```
@Entity
@Table(name="Instructor")
public class Instructor {
```

```
@Id
@GeneratedValue(strategy=GenerationType.SEQUENCE)
@Column(name="Id")
private int id;

@Column(name="Instructor_Name")
private String name;
```

- @Entity : Specifies that the class is an entity. This annotation can be applied on Class, Interface or Enums.
- @Table : It specifies the table in the database with which this entity is mapped.
- @Column : Specify the column mapping using @Column annotation. Use it to override default values
 - Updatable = false/true, name="MyNewColumnName" , nullable= false, length = 50
- @ID and @GeneratedValue: This annotation specifies the primary key of the entity and Generated value specifies the generation strategies for the values of primary keys.
 - JPA strategies
 - AUTO, TABLE , IDENTITY , SEQUENCE.
- @OrderBy : Sort your data using @OrderBy annotation.

Hibernate CURD API

- `Session.save(entity reference);`
- `Session.get(EntityClass,ID);`
- `Session.update(entity reference);`
- `Session.delete(entity reference)`

Controlling Primary Key

- @GeneratedValue annotation, which the increase type for auto increment column.
- @SequenceGenerator annotation is used to govern auto increment behaviour.

```
@Id
@SequenceGenerator(name="mylogic",initialValue=1100,allocationSize=100)
@GeneratedValue(strategy=GenerationType.SEQUENCE,generator="mylogic")
private int myid;
```


Assignment

- Create a Hibernate specific application to store the information of the client.

BankUser
String : emailID (as primary key) String : name (cannot be null) int : balance String : address (length : cannot be more than 100) int passport : (unique)

- Perform all CRUD operations on the record, using Hibernate CRUD API.

Understanding Object Type

```
@Entity
@Table(name="Account")
public class Account {

    @Embedded
    private Address address;
```

```
import javax.persistence.Embeddable;
@Embeddable
public class Address {

    private String houseAddress;
    private String city;
    private String country;
```

- Type of Objects.
 - Value Objects are the objects which can not stand alone.
 - Entity Objects are those who can stand alone like College and Student.
- When we define the value object for the entity class we use @Embeddable.
- When we use value type object in entity class we use @Embedded.
- Use @AttributeOverrides to override the column information

Saving Collection Objects

- Need to store the information in separate table

```
@Entity
@Table(name="Account")
public class Account {

    @ElementCollection
    @CollectionTable(name="Policy_Account",joinColumns=@JoinColumn(name="LinkedAccount"))
    private List<Policy> policies = new ArrayList<Policy>();
```

```
@Embeddable
public class Policy {

    private int policyNumber;
    private String policyName;
    private int balance;
```

Assignment

- For the below classes , save the information in the database.
- For Account Table primary key should be started from 1000 and incremented by 10
- And for collection table , foreign key name must be “linkedAccount” and table name must be “Account_Policy”.
- Policy table also has auto incremented primary key through `@GenericGenerator`.
- `Read all policies related to Account`

Account

```
long panNumber;  
long phoneNumber  
String name  
int balance ;  
  
List<Policy> policies;
```

Policy

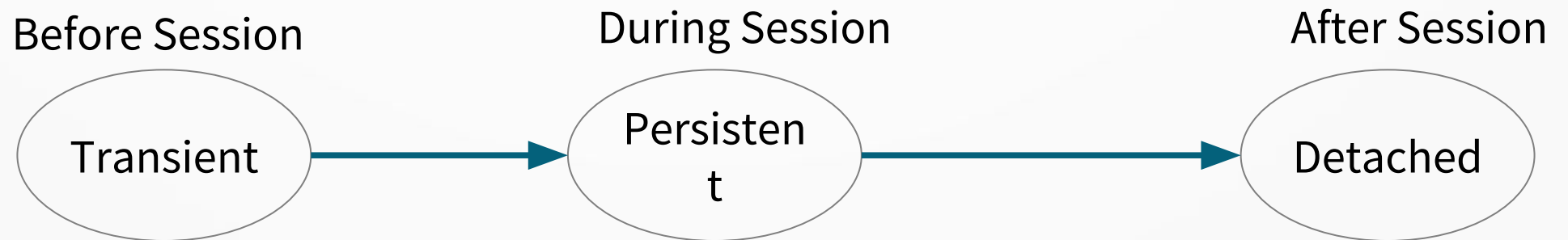
```
int policyNumber;  
String policyName;  
int policyAmount;
```

Hibernate and JPA API

- We will learn Session State.
- will learn also Session methods.
 - Save and Persist
 - Get and load.
 - Update and saveorupdate.
- Lazy loading and Eager loading
- Session methods like `evict()` , `clear()`, and `Close()`.

Hibernate Session State

- Hibernate has provided three states of an Object.
- These three stages often known as Object Lifecycle states
 - Transient State.
 - Persistent State.
 - Detached State.



Transient State

- A New instance of a persistent class which is not associated with a Session.
- has no representation in the database and no identifier value is considered transient by Hibernate.

Persistent state

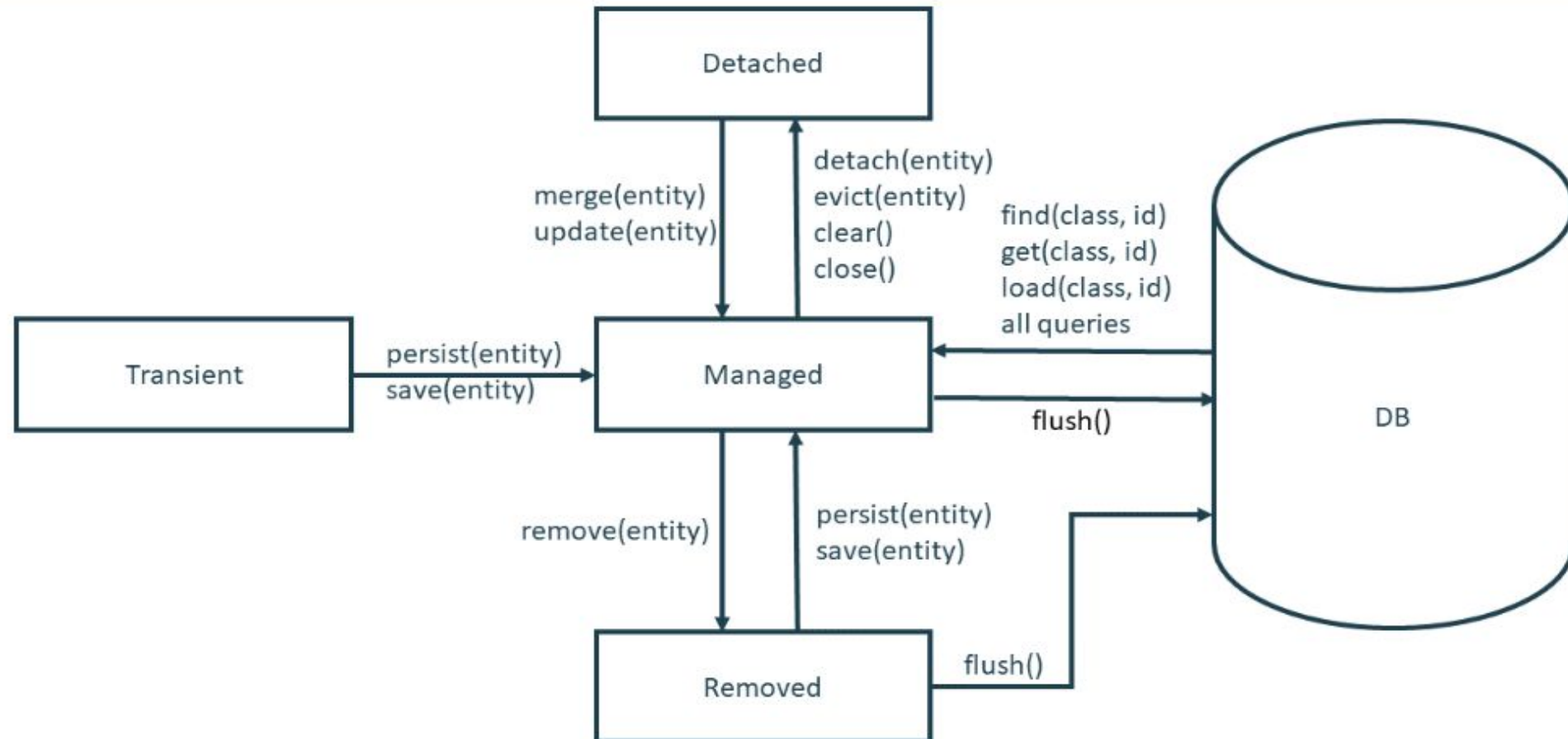
- A persistent instance has a representation in the database .
- An identifier value and is associated with a Session.
- We can make a transient instance persistent by associating it with a Session.
- The Persistent object represents the database entity and its having an unique identifier value, which represents the database table.
- The values associated with the persistent object are sync with the database.
- It means, if we change the values in persistent state objects, the changes will automatically get effected in the database, no need to execute insert/update operations.

Detached State

- if we close the Hibernate Session, the persistent instance will become a detached instance.
- it isn't attached to a Session anymore.
- Session evict() is used to make the associated object as Detached.
- Session.merge() is used to re-associate the object with session.

Session API

- JPA and Hibernate provide different methods to persist new and to update existing entities.
- You can use the methods persist and save to store a new entity and the methods merge and update to store the changes of a detached entity in the database.



Save or persist

- The save method, on the other hand, is Hibernate-specific. It is, therefore, not available in other JPA implementations.
- JPA's persist method returns void and Hibernate's save method returns the primary key of the entity.
- Also trying to save the persistent object again leads to the duplication of entry in case of save , whereas in case of persist it leads to an exception.

Get and load()

- Similar to save and persist , get is Hibernate specific method where as load is JPA implementation.
- Get makes the call of select statement at a time of call.
- Load makes the call of select statement at a time of persistent object get method is called meanwhile it works proxy object.

Evict() and merge()

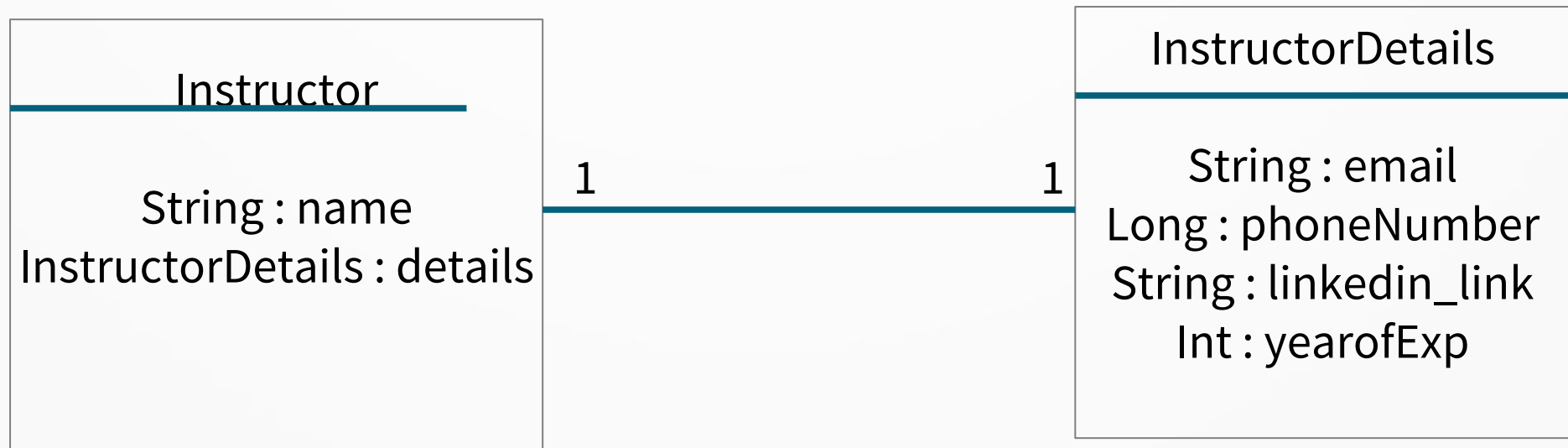
- Session.evict() is used to detached the specific object.
- Merge is used to re-associate the same object with session again.
- Session also has following methods which leads peristent object to detached state.
 - Session.flush()
 - Session.clear()
 - Session.close()

Entity Mapping

- Hibernate is best known as for its Mapping.
- It is the procedure to convert relational model into Java object model.
- Such as implementing relationships between entities.
 - One to one
 - One to many and
 - Many to many.
- Hibernate provides more than the entity relationship model using its bi-directional strategy to implement.

One to One Mapping

- One to One relation specifies that an entity(A) is associated to only a single instance of another entity(B). From database perspective, you can assume that if table A has a one to one mapping with table B, then each row of Table B will have a foreign key column which refers to a primary key of Table A.



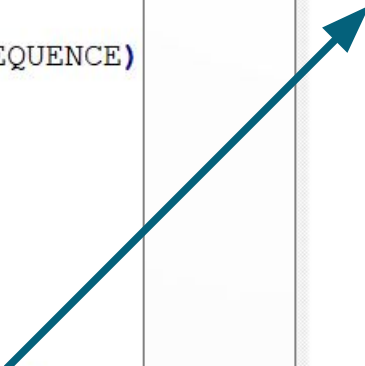
Class Structure

```
@Entity
@Table(name="Instructor")
public class Instructor {

    @Id
    @GeneratedValue(strategy=GenerationType.SEQUENCE)
    @Column(name="Id")
    private int id;

    @Column(name="Instructor_Name")
    private String name;

    @OneToOne(cascade=CascadeType.ALL)
    @JoinColumn(name="instructor_detail_id")
    private InstructorDetails instructorDetails;
```



```
@Entity
@Table(name="InstructorDetails")
public class InstructorDetails {

    @Id
    @GeneratedValue(strategy=GenerationType.SEQUENCE)
    @Column(name="id")
    private int id;
    @Column(name="Instructor_location")
    private String location;
    @Column(name="Instructor_LinkedIn")
    private String linkedIn;
```

Use @oneToOne annotation on the field which holds the relationship

And @JoinColumn marks a column for as a join column for an entity association or an element collection.

OnetoOne bi directional

```
@Entity
@Table(name="Instructor")
public class Instructor {
```

```
    @Id
    @GeneratedValue(strategy=GenerationType.SEQUENCE)
    @Column(name="Id")
    private int id;
```

```
    @Column(name="Instructor_Name")
    private String name;
```

```
    @OneToOne(cascade=CascadeType.ALL)
    @JoinColumn(name="instructor_detail_id")
    private InstructorDetails instructorDetails;
```

mappedBy



```
@Entity
@Table(name="InstructorDetails")
public class InstructorDetails {
```

```
    @Id
    @GeneratedValue(strategy=GenerationType.SEQUENCE)
    @Column(name="id")
    private int id;
    @Column(name="Instructor_location")
    private String location;
    @Column(name="Instructor_LinkedIn")
    private String linkedIn;
```

```
    @OneToOne(mappedBy="instructorDetails", cascade=CascadeType.ALL)
    private Instructor instructor;
```


One to many

```
@Entity
@Table(name="Instructor")
public class Instructor {

    @Id
    @GeneratedValue(strategy=GenerationType.SEQUENCE)
    @Column(name="Id")
    private int id;

    @Column(name="Instructor_Name")
    private String name;



    @Column(name="Instructor_email")
    private String email;







    /* One to many code starts */

    @OneToMany(cascade=CascadeType.ALL)
    @JoinColumn(name="Trainer")
    private Set<Course> courses;
```

```
@Entity
@Table(name="Course")
public class Course {

    @Id
    @GeneratedValue(strategy=GenerationType.SEQUENCE)
    private int id;
    private String courseName;
```

EDIT	ID	INSTRUCTOR_EMAIL	INSTRUCTOR_NAME
	1	saurabh@gmail.com	saurabh
	6	ashish@gmail.com	Ashish
row(s) 1 - 2 of 2			

EDIT	ID	COURSENAME	TRAINER
	2	JAX-RS	1
	3	Spring	1
	4	Core Java	6
	5	React	1
	7	SFDC	6
	8	Servlet and JSP	6
	9	Hibernate	6
row(s) 1 - 7 of 7			

One to many bi Directional

```
@Entity
@Table(name="Instructor")
public class Instructor {

    @Id
    @GeneratedValue(strategy=GenerationType.SEQUENCE)
    @Column(name="Id")
    private int id;

    @Column(name="Instructor_Name")
    private String name;

    @Column(name="Instructor_email")
    private String email;

    /* One to many code starts */

    @OneToMany(cascade=CascadeType.ALL)
    @JoinColumn(name="Trainer")
    private Set<Course> courses;
```

```
@Entity
@Table(name="Course")
public class Course {

    @Id
    @GeneratedValue(strategy=GenerationType.SEQUENCE)
    private int id;
    private String courseName;

    @ManyToOne
    @JoinColumn(name="Trainer")
    private Instructor instructor;
```

@ManyToOne and Joining the Column with The same column of parent entity will leads to OneToMany bi directional mapping

ManyToMany

- Many to many relationship will always leads to the creation of three tables.
 - Two for separate entity . And
 - Third for relationship

@ManyToMany

```
@Entity
@Table(name="Hotels")
public class Hotels {
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private int id;
    private String hotelName;

    @ManyToMany(cascade=CascadeType.ALL)
    @JoinTable(
        name="TouristApp_Hotels",
        joinColumns=@JoinColumn(name="Hotel_ID"),
        inverseJoinColumns=@JoinColumn(name="TouristApp"))
    private List<TouristApp> listedApps;
```

```
@Entity
@Table(name="TouristApp")
public class TouristApp {






    @Id
    private String appName;
    private int activeUserCount;

    @ManyToMany(cascade=CascadeType.ALL)
    @JoinTable(
        name="TouristApp_Hotels",
        joinColumns=@JoinColumn(name="TouristApp"),
        inverseJoinColumns=@JoinColumn(name="Hotel_ID"))
    private List<Hotels> hotelList;
```




Hotels Table

EDIT	ID	HOTELNAME
	1	Hotel1
	2	Hotel2
	3	Hotel4
	4	Hotel3
	5	Hotel5
row(s) 1 - 5 of 5		

And Relationship table
TouristApp_Hotels

EDIT	TOURISTAPP	HOTEL_ID
	TApp1	1
	TApp1	2
	TApp1	4
	TApp1	3
	TApp1	5

TouristApp
Table

EDIT	APPNAME	ACTIVEUSERCOUNT
	TApp1	22
	Tapp2	44
	Tapp3	44
row(s) 1 - 3 of 3		

Inheritance mapping

- Relational databases don't have a straightforward way to map class hierarchies onto database tables.
- Because such type of relationship does not available in relational domain.
- Hibernate provides three strategies to implement such relationship.
 - Single Table
 - Table Per Class
 - Joined

Single Table.

```
@Entity
@Table(name="Car")
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
public class Car {

    @Id
    @GeneratedValue(strategy=GenerationType.SEQUENCE)
    int id;




    String brand;
```

```
@Entity
public class HatchBack extends Car {

    String space;
```

```
@Entity
public class SportsCar extends Car{

    String speed;
```

EDIT	DTYPE	ID	BRAND	SPACE	SPEED
	Car	1	Maruti	-	-
	SportsCar	2	Farari	-	200
	HatchBack	3	Hyndai	5	-
row(s) 1 - 3 of 3					

```
@Entity
@Table(name="Car")
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="Car_Type",discriminatorType=DiscriminatorType.STRING)
public class Car {

    @Id
    @GeneratedValue(strategy=GenerationType.SEQUENCE)
    int id;
```






EDIT	CAR_TYPE	ID	BRAND	SPACE	SPEED
	Car	1	Maruti	-	-
	SportsCar	2	Farari	-	200
	HatchBack	3	Hyndai	5	-
row(s) 1 - 3 of 3					




Table Per Class and Joined


- Table per and Joined startegy creates tables per @Entity class.


EDIT	ID	BRAND
	1	Maruti
row(s) 1 - 1 of 1		

EDIT	ID	BRAND	SPACE
	3	Hyundai	5
row(s) 1 - 1 of 1			

EDIT	ID	BRAND	SPEED
	2	Farari	200
row(s) 1 - 1 of 1			

EDIT	ID	BRAND
	1	Maruti
	2	Farari
	3	Hyndai
row(s) 1 - 3 of 3		

EDIT	SPACE	ID
	5	3
row(s) 1 - 1 of 1		

EDIT	SPEED	ID
	200	2
row(s) 1 - 1 of 1		

HQL

HQL is the Hibernate Query Language , used by the hibernate to query from database.

HQL is the database independent language (opposite to SQL).

```
import org.hibernate.query.Query;  
Query<Student> q = session.createQuery("From Student");  
List<Student> list = q.list();
```

More HQL Queries

- 1) `select marks,address from Student`
- 2) `String hql = select marks,address from Student where id = ?;`
`Query<Student> query = Session.createQuery(hql,Student.class);`
`query.setParameter(0,stuID);`
`Student student = query.uniqueResult(); // we can also use list()`
- 3) `String hql = select marks,address from Student where id : filterid;`
`Query<Student> query = Session.createQuery(hql,Student.class);`
`query.setParameter("filterid",stuID);`
`Student student = query.uniqueResult(); // we can also use list()`

HQL Other operation

```
Query<Student> q = session.createQuery("update Student set address = :newaddress where id =  
:filterid");  
q.setParameter("newaddress", "Updated Address - 123");  
q.setParameter("filterid", 1);  
  
Transaction t = session.beginTransaction();  
  
int id = q.executeUpdate();  
  
t.commit();
```


Caching

- Hibernate Caching Mechanism is to improve the application performance.
- Hibernate contains two level caches. It will be placed between the application and database, They are Level1 cache and Level2 cache.
- Level1 cache is also called Session level Cache which maintains session object. When a session is opened then the cache opens automatically.

Continue...

- If one performs any Database operations then they can create a session object. Session connection is automatically opened and closed in Hibernate using `get()`. It goes to Level1 cache and checks for objects, if an object already exists in Database, then it reads the object from Level1 cache else, reads objects from database and stores it in Level1 cache, and then reads the object from the database.
- Session object can be created N-number of times. But, it can't maintain entire application. With the help of cache, hibernate will improve the performance of an application by reducing the number of round trips between application and database. From a level1 cache, one can remove a particular object and can call `evict()` of session interface.

Level 2 Cache

- Level2 cache object maintains SessionFactory.
- This object contains entire application. But, it should be enabled explicitly.
- Level2 Cache implementations are provided by different vendors such as.
 - OSCache
 - EH Cache
 - JBOSS Cache

Implementation – maven jars

```
<!-- hibernate-core -->
```

```
<dependency>  
<groupId>org.hibernate</groupId>  
<artifactId>hibernate-core</artifactId>  
<version>${hibernate.version}</version>  
</dependency>
```

```
<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-ehcache -->
```

```
<dependency>  
<groupId>org.hibernate</groupId>  
<artifactId>hibernate-ehcache</artifactId>  
<version>5.2.12.Final</version>  
</dependency>
```

```
<!-- https://mvnrepository.com/artifact/org.ehcache/ehcache -->
```

```
<dependency>  
<groupId>org.ehcache</groupId>  
<artifactId>ehcache</artifactId>  
<version>3.4.0</version>
```

Cfg.xml and Entity Class

```
<property name="hibernate.cache.use_second_level_cache">true</property>
<property
name="hibernate.cache.region.factory_class">org.hibernate.cache.ehcache.EhCacheRegionFactory</prop
erty>
<property
name="hibernate.javax.cache.provider">org.hibernate.cache.ehcache.EhCacheRegionFactory</property>
```

```
@Entity    // HQL
@Table(name="MyStudents")
@Cache(usage=CacheConcurrencyStrategy.READ_WRITE,
region="MyStudents")
public class Student {
    ....
}
```

MKJ IT Solutions

A Corporate Training Company



Corporate Trainings



Communication Training



Skill Development