Hibernate is since 2001, hibernate existed before JPA, and since JPA was modelled on Hibernate,

## Hibernate 4,5,6

|  |  |
| --- | --- |
| In hibernate 4 | In hibernate 5 onwards |
| Old way to create session factory using configuration obj | New way to create session factory using StandardServiceRegistry |
| session.save()is deprecated | Instead use ses.persist() |
| Old is from javax.persistence | For hib 6, java 11 Is mandatory  From hib 6+ All import statements from Jakarta.persistence |
| For primary key auto generation use  @Id   @GeneratedValue(strategy = GenerationType.AUTO) | @Id  @GeneratedValue(strategy = GenerationType.IDENTITY) |
|  |  |
|  |  |
|  |  |

References

or goto user guide in below link

<https://hibernate.org/orm/documentation/6.4/>

Main Reference documentation user guide <https://docs.jboss.org/hibernate/orm/6.4/userguide/html_single/Hibernate_User_Guide.html>

Intro to hib <https://docs.jboss.org/hibernate/orm/6.4/introduction/html_single/Hibernate_Introduction.html>

HQL guide <https://docs.jboss.org/hibernate/orm/6.4/querylanguage/html_single/Hibernate_Query_Language.html>

For data modelling <https://agiledata.org/essays/dataModeling101.html>

New users may want to first look at the tutorial-style [Quick Start](https://docs.jboss.org/hibernate/orm/6.4/quickstart/html_single/)  <https://docs.jboss.org/hibernate/orm/6.4/quickstart/html_single/> guide.

This User Guide is really more of a reference guide. For a more high-level discussion of the most used features of Hibernate, see the [Introduction to Hibernate](https://docs.jboss.org/hibernate/orm/6.4/introduction/html_single/Hibernate_Introduction.html) guide.

There is also a series of [topical guides](https://docs.jboss.org/hibernate/orm/6.4/topical/html_single/) providing deep dives into various topics such as logging, compatibility and support, etc.

<https://docs.jboss.org/hibernate/orm/6.4/topical/html_single/>

<https://github.com/hibernate>

**Old way**

Configuration cfg = new Configuration**()**.addResource**(**"Employee.hbm.xml"**)**.addResource**(**"Address.hbm.xml"**)**;

SessionFactory sessionFactory = cfg.buildSessionFactory**()**;

Session session = sessionFactory.openSession**()**;

**New way to create session factory object**

private static StandardServiceRegistry registry;

private static SessionFactory sessionFactory;

public static SessionFactory getSessionFactory() {

if (sessionFactory == null) {

try {

// Create registry

registry = new StandardServiceRegistryBuilder().configure().build();

// Create MetadataSources

MetadataSources sources = new MetadataSources(registry);

// Create Metadata

Metadata metadata = sources.getMetadataBuilder().build();

// Create SessionFactory

sessionFactory = metadata.getSessionFactoryBuilder().build();

Maven dependencies - BOM

Hibernate also provides a platform (BOM in Maven terminology) module which can be used to align versions of the Hibernate modules along with the versions of its libraries. The platform artifact is named hibernate-platform.

To apply the platform in Gradle

dependencies **{**

implementation platform "org.hibernate.orm:hibernate-platform:6.4.4.Final"

*// use the versions from the platform*

implementation "org.hibernate.orm:hibernate-core"

implementation "jakarta.transaction:jakarta.transaction-api"

**}**

See the [Gradle documentation](https://docs.gradle.org/current/userguide/java_platform_plugin.html#sec:java_platform_consumption) for capabilities of applying a platform.

To apply the platform (BOM) in Maven

<dependency>

<groupId>org.hibernate.orm</groupId>

<artifactId>hibernate-core</artifactId>

</dependency>

<dependency>

<groupId>jakarta.transaction</groupId>

<artifactId>jakarta.transaction-api</artifactId>

</dependency>

<dependencyManagement>

<dependency>

<groupId>org.hibernate.orm</groupId>

<artifactId>hibernate-platform</artifactId>

<version>6.4.4.Final</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencyManagement>

Bootstrapping hibernate code

<https://docs.jboss.org/hibernate/orm/6.4/quickstart/html_single/>

the below code is from above website

**protected** **void** **setUp()** **{**

*// A SessionFactory is set up once for an application!*

**final** **StandardServiceRegistry** registry **=**

**new** **StandardServiceRegistryBuilder()**

**.**build**();**

**try** **{**

sessionFactory **=**

**new** **MetadataSources(**registry**)**

**.**addAnnotatedClass**(Event.**class**)**

**.**buildMetadata**()**

**.**buildSessionFactory**();**

**}**

**catch** **(Exception** e**)** **{**

*// The registry would be destroyed by the SessionFactory, but we*

*// had trouble building the SessionFactory so destroy it manually.*

**StandardServiceRegistryBuilder.**destroy**(**registry**);**

**}**

**}**

Terminologies

Javax 🡪 EntityManagerFactory (Interface dev by javax)

|

|

SessionFactory (interface extended by hibernate)

Same like above session is child extending EntityManager

As an application developer, you must decide whether to:

* write your program in terms of Session and SessionFactory, or
* maximize portability to code in other implementations of JPA by, wherever reasonable, writing terms of EntityManager and EntityManagerFactory, falling back to the native APIs only where necessary (try to prefer jpa over hibernate for future switch from hib to another vendor, try to avoid using vendor specific anno).
* But in real time we will mostly use spring
* But without the native APIs, and extended mapping annotations, you miss out on much of the power of Hibernate

Whichever path you take, you will use the JPA-defined mapping annotations most of the time, and the Hibernate-defined annotations for more advanced mapping problems.

Saving using JPA vs hib orentity managers vs session

The EntityManagerFactory is the Jakarta Persistence equivalent of a SessionFactory

// u can perform save operation using both session and entity manager

Saving using hibernate

sessionFactory**.**inTransaction**(**session **->** **{**

session**.**persist**(new** **Event(**"Our very first event!"**,** now**()));**

session**.**persist**(new** **Event(**"A follow up event"**,** now**()));**

**});**

sessionFactory**.**inTransaction**(**session **->** **{**

session**.**createSelectionQuery**(**"from Event"**,** **Event.**class**)**

**.**getResultList**()**

**.**forEach**(**event **->** out**.**println**(**"Event (" **+** event**.**getDate**()** **+** ") : " **+** event**.**getTitle**()));**

**});**

Full program saving and fetching using hibernate

**package** org.hibernate.example**;**

**import** org.hibernate.cfg.Configuration**;**

**import** static java**.**lang**.**Boolean**.**TRUE**;**

**import** static java**.**lang**.**System**.**out**;**

**import** static org**.**hibernate**.**cfg**.**AvailableSettings**.\*;**

**public** **class** **Main** **{**

**public** **static** **void** **main(String[]** args**)** **{**

**var** sessionFactory **=** **new** **Configuration()**

**.**addAnnotatedClass**(Book.**class**)**

*// use H2 in-memory database*

**.**setProperty**(**URL**,** "jdbc:h2:mem:db1"**)**

**.**setProperty**(**USER**,** "sa"**)**

**.**setProperty**(**PASS**,** ""**)**

*// use Agroal connection pool*

**.**setProperty**(**"hibernate.agroal.maxSize"**,** "20"**)**

*// display SQL in console*

**.**setProperty**(**SHOW\_SQL**,** TRUE**.**toString**())**

**.**setProperty**(**FORMAT\_SQL**,** TRUE**.**toString**())**

**.**setProperty**(**HIGHLIGHT\_SQL**,** TRUE**.**toString**())**

**.**buildSessionFactory**();**

*// export the inferred database schema*

sessionFactory**.**getSchemaManager**().**exportMappedObjects**(true);**

*// persist an entity*

sessionFactory**.**inTransaction**(**session **->** **{**

session**.**persist**(new** **Book(**"9781932394153"**,** "Hibernate in Action"**));**

**});**

*// query data using HQL*

sessionFactory**.**inSession**(**session **->** **{**

out**.**println**(**session**.**createSelectionQuery**(**"select isbn||': '||title from Book"**).**getSingleResult**());**

**});**

*// query data using criteria API*

sessionFactory**.**inSession**(**session **->** **{**

**var** builder **=** sessionFactory**.**getCriteriaBuilder**();**

**var** query **=** builder**.**createQuery**(String.**class**);**

**var** book **=** query**.**from**(Book.**class**);**

query**.**select**(**builder**.**concat**(**builder**.**concat**(**book**.**get**(**Book\_**.**isbn**),** builder**.**literal**(**": "**)),**

book**.**get**(**Book\_**.**title**)));**

out**.**println**(**session**.**createSelectionQuery**(**query**).**getSingleResult**());**

**});**

**}**

**}**

Saving using Jpa

**protected** **void** **setUp()** **{**

entityManagerFactory **=** **Persistence.**createEntityManagerFactory**(**"org.hibernate.tutorial.jpa"**);**

**}**

**void** **inTransaction(Consumer<EntityManager>** work**)** **{**

**EntityManager** entityManager **=** entityManagerFactory**.**createEntityManager**();**

**EntityTransaction** transaction **=** entityManager**.**getTransaction**();**

**try** **{**

transaction**.**begin**();**

work**.**accept**(**entityManager**);**

transaction**.**commit**();**

**}**

**catch** **(Exception** e**)** **{**

**if** **(**transaction**.**isActive**())** **{**

transaction**.**rollback**();**

**}**

**throw** e**;**

**}**

**finally** **{**

entityManager**.**close**();**

**}**

**}**

Full program saving and fetching using jpa

**package** org.hibernate.example**;**

**import** jakarta.persistence.EntityManager**;**

**import** jakarta.persistence.EntityManagerFactory**;**

**import** java.util.Map**;**

**import** java.util.function.Consumer**;**

**import** static jakarta**.**persistence**.**Persistence**.**createEntityManagerFactory**;**

**import** static java**.**lang**.**System**.**out**;**

**import** static org**.**hibernate**.**cfg**.**AvailableSettings**.**JAKARTA\_HBM2DDL\_DATABASE\_ACTION**;**

**import** static org**.**hibernate**.**tool**.**schema**.**Action**.**CREATE**;**

**public** **class** **Main** **{**

**public** **static** **void** **main(String[]** args**)** **{**

**var** factory **=** createEntityManagerFactory**(**"example"**,**

*// export the inferred database schema*

**Map.**of**(**JAKARTA\_HBM2DDL\_DATABASE\_ACTION**,** CREATE**));**

*// persist an entity*

inSession**(**factory**,** entityManager **->** **{**

entityManager**.**persist**(new** **Book(**"9781932394153"**,** "Hibernate in Action"**));**

**});**

*// query data using HQL*

inSession**(**factory**,** entityManager **->** **{**

out**.**println**(**entityManager**.**createQuery**(**"select isbn||': '||title from Book"**).**getSingleResult**());**

**});**

*// query data using criteria API*

inSession**(**factory**,** entityManager **->** **{**

**var** builder **=** factory**.**getCriteriaBuilder**();**

**var** query **=** builder**.**createQuery**(String.**class**);**

**var** book **=** query**.**from**(Book.**class**);**

query**.**select**(**builder**.**concat**(**builder**.**concat**(**book**.**get**(**Book\_**.**isbn**),** builder**.**literal**(**": "**)),**

book**.**get**(**Book\_**.**title**)));**

out**.**println**(**entityManager**.**createQuery**(**query**).**getSingleResult**());**

**});**

**}**

*// do some work in a session, performing correct transaction management*

**static** **void** **inSession(EntityManagerFactory** factory**,** **Consumer<EntityManager>** work**)** **{**

**var** entityManager **=** factory**.**createEntityManager**();**

**var** transaction **=** entityManager**.**getTransaction**();**

**try** **{**

transaction**.**begin**();**

work**.**accept**(**entityManager**);**

transaction**.**commit**();**

**}**

**catch** **(Exception** e**)** **{**

**if** **(**transaction**.**isActive**())** transaction**.**rollback**();**

**throw** e**;**

**}**

**finally** **{**

entityManager**.**close**();**

**}**

**}**

**}**

# Session operations

Session object = connection Object ++

L1 cache= session level cache

L2 cache= session factory level cache

Composition vs aggregation

Aggregation- means a group of houses in colony- there is no tight relationship among themselves,

Composition:-means there is a tight relationship- if house is destroyed all rooms are destroyed

Ses.close() will closes the connection and gives that conn obj back to conn pool

### Ses.save()

ses.save() or ses.persist() // when we are executing these statements in transaction, these will not insert rec in db… need to check

actual insertion happens only when txn.commit() is issued

### Ses.persist() example

Session s = sf.openSession();  
Transaction transaction = null;  
if(!s.getTransaction().isActive()){  
 System.*out*.println("current transaction is not active so creating new ");  
 transaction = s.beginTransaction();  
}  
s.persist(p);  
transaction.commit();

# Mappings Associations

Always think like why not many to many

While fetching we have to think about Eager or lazy loading

and while inserting/updating/deletion we have to think about cascading

|  |  |
| --- | --- |
|  |  |
|  |  |
| Many to One – Many employees can belong to same dept  Means each employee belongs to one dept | Class Employee{  @ManyToOne  Department dt;// no need of list  } |
| Many to Many | From both sides one to many  Patient doctor, programmer project  In case of many to many we have to store mappings in separate table |

Mapping Strategies

1. Write as 2 pojos(only 1 can be entity) but store in single table –means fields in 2 class must be stored in single table and they are not in relationship also, we can achieve this using @Embeddable, @Embedded annotations

Like , Address(normal pojo), Employee(Entity) both these classes are separate but store them in single table

Class Employee {

@Embedded

Address add;

}

@Embedabble

Class Address

Refer 81st class notes



1. When to go for inheritance:- IS-A relationship- if one class wants to use the entire properties of another class then go for inheritance
2. When to go for has-A relationship- if one class wants to use only some properties of another class then go for

HAS-A relationship

1. Always ensure to keep common data in another table and refer using relationships

Note:- in annotation models there are only 3 strategies- TPSC,TPCC,TPC(Table per class)

In anno TPCC with implicit or explicit polymorphism is not possible with default setup only TPCC is possible

In xml we have 1 extra strategy- TPCC table per concrete class with implicit polymorphism

1. **TPSC- Table per sub class -**If entities are in IS-A relationship parent-child relationship then go for 1 separate table for each child, one separate table for parent (Best is Table per sub class -TPSC) because data will be in both parent and child, so no duplicate data as parent columns will not be in child table hence this is industry standard

**Main advantage:**- is if u insert in child table record automatically inserts in parent table



1. TPCC- table per concrete class-no table for parent class-tables only for child class –

Problem is parent class attributes will be present in all child’s leads to duplicate fields in all tables

This is Worst is - as it leads to presence of same duplicates columns in all child tables

Ex:- parent (payment- has 2 fields named – payment id, payment amount)

Childs – card payment, cheque payment so now in TPCC parent class 2 fields are present in all childs

If u want those duplicates in all childs then its ur wish

Here use annotations like @Inheritance (strategy=InheritenceType.joined) on super class

This annotation will be kept on child table to indicate this table will be joined with another table which has primary key

@PrimarykeyJoinColumn(name=”payment\_id”,referencedColumnName=”pid”)

|  |  |
| --- | --- |
| @DiscriminatorCoumn(name=”payment\_mode”) –in parent table  @Inheritance (strategy=InheritenceType.joined) – in parent cls  Means all tables are joined |  |
| @DiscriminatorValue(“CARD”) –in child table | @DiscriminatorValue(“PAYMENT”) |

but if parent has 2 childs, because of while child table record the record inserted into parent table if u want to know that

then u have to keep both above annotations, when child record is created in child1 that same value is inserted in parent



OneToMany

Mapped By/avoid extra table for mapping

**Refer paint brush 92**

Mapped by means, every row in child table is mapped to a parent record in parent table

|  |  |  |
| --- | --- | --- |
| Lap\_id | Laptop name | employeeId |
|  |  |  |
|  |  |  |
|  |  |  |

@Entity  
public class Employee {  
 @OneToMany(**mappedBy= “”**)  
 List<Laptop> laptops;  
   
 @Id  
 @GeneratedValue  
 private Long id;  
  
 String name;  
  
 Double marks;  
  
}

If u don’t specify that mappedBy then a separate table will be created with mappings

One Employee have many Laptops, since in Employee tables as we cant have a list type,

Internally hibernate will create a separate table,

|  |  |
| --- | --- |
| Student\_id | Laptop\_id |
| S1 | L1 |
| S1 | L2 |

The above states, each student will have many laptops (s1 having 2 laps called l1,l2)

To stop this additional tables, we have to tell hibernate saying, BRO don’t create another table

Instead in laptop column itself we will create employee , use that , then we have to tell mapped by field

@OneToMany(mappedBy = "employee")  
List<Laptop> laptops;

## Mapping statements

1. One category will have many products & Many products belongs same category (here think in database row level)
2. One dept will have many student, each student belongs to one dept

## Cascade types

Any non select(insert,update,delete) operation performed on parent objects will be performed on child objects too

What about insertion in child?if we insert in child will it be reflected in parent?

CascadeType.aLL- means when we save Product , then save dependent Category object also to db

Ex:- if u insert parent object, associated child objects will also be inserted,

if u delete parent object associated child objects also will be deleted

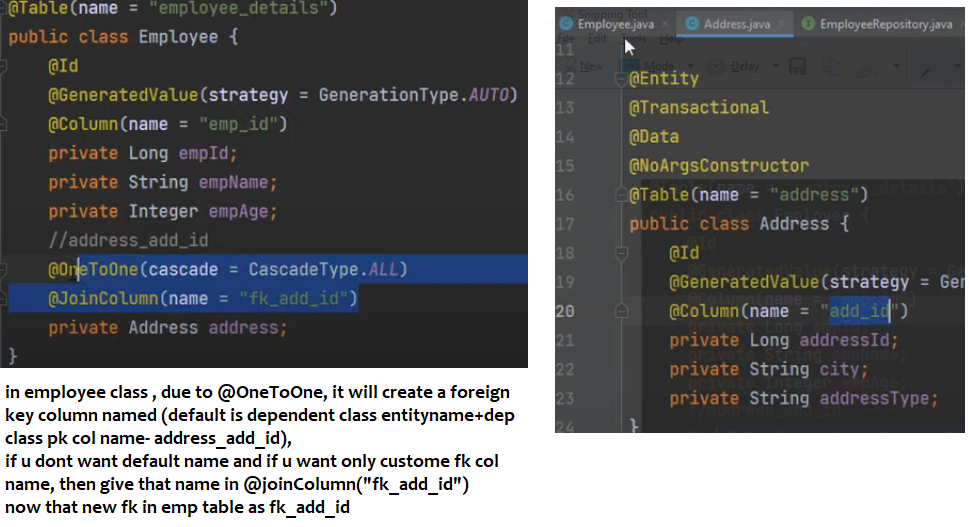
public class Product {  
 @ManyToOne(cascade = CascadeType.*ALL*)  
 @JoinColumn(name = "category\_id")  
 private Category category;

@JoinColumn

**in employee class , due to @OneToOne, it will create a foreign key column named (default is dependent class entityname+dep class pk col name- address\_add\_id),**

**if u dont want default name and if u want only custome fk col name, then give that name in @joinColumn("fk\_add\_id")**

**now that new fk in emp table as fk\_add\_id**



Cascade Delete

Refer proj 68

**Always be cautious while deleting, if u enable cascade=All**

Remember we should delete parent record only when u deleted all childs

Assume 1 parent will have 3 childs, in cascade=all if u delete any 1 child parent also should be deleted

if u delete 2nd child due to cascade=Cascade.All it will try to delete parent it will throw exceptions as that parent having still 2 more childs

Synchronization from session cache to db

Ses.beginTransaction()

Department dept = ses.get(Department.class, 12);//all the parent and childs will be loaded into session cache  
List<Employee> list = dept.getEmployeesList();

List.clear() // if u delete the list all childs from session cache will be deleted and synchronization will happen to db on commit

tx.commit();

Cascade insert

To work with cascade insert mappedBy is mandatory and u have to set the parent to each and every child and set

The child to each and every parent

#### Lazy loading

**Lazy loading means- lazily loading the associated child objects**

**If u enable lazy loading then we must fetch child obj or parent obj in using ses.load()/ ses.get() in transaction only**

**Lazy loading is the default fetch strategy**

**if we load parent object child objects will be not fetched untill we use the related methods**

**in above, from parent object we used only parent class fields like useid , username,address**

**hence at this point childs will not be fetched and its called lazy loading**

**refer screenshot92**

**in last line when we are priniting the childs info then it will assume we are using child data , at that line only it will go db and hit and get the records hence it is called lazy loading**

why lazy loading needs transaction

**When we are using 1-many relationship, while fetching parent, childs records will not be fetched until we use child objects, if we use child objects by call child fields like in println then it will be fetched**

So here 2 queries will be used to fetch parent and child, since 2 queries are there txn is mandatory

Lazy loading makes hibernate app hitting the database only when need is there, it delays the process of hitting as long as it can

**Lazy loading in Many to One**

**---------------------------------**

**Class Employee{**

**String employeeName;**

**@ManyToOne(fetch=FetchType.Lazy)**

**Department dept;**

**}**

**//here in Many to One (Many employees belong to single department) first all childs will be loaded and then**

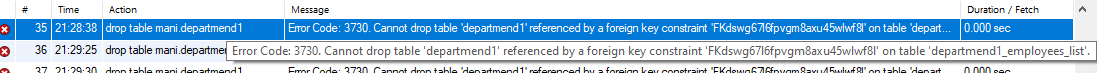
**if u use parent Dept object then only a query will be generated to fetch parent Dept info which is being loaded lazily which is called lazy loading**

## Delete operations

Refer prj-68 for delete operations

We cannot Delete parents without deleting childs, 1st u have to delete childs then u can delete parents

Ex:- wife husband divorced, what abt kids, hence 1st court will ask what about kids, 1st settle kids then think of divorce



When u are deleting parent,

Always prefer loading and delete using ses.load, then it will load all parents and childs into session object

Don’t delete parent using HQL, if u use HQL to delete parent, it will try to delete parent alone not childs

Whereas ses.load() will load all the parent and child

Due to cascadeType.Delete Once u deleted parent, childs will also be deleted automatically