Hibernate

What is Persistence?

Data remains stored and accessible even if the **application stops,** example in amazon or netflix .. persistence storage like (DB). Or storing Data for a long time..

Now to perform these persistence logic and operations we need a **persistence Framework** or a tool which is Hibernate !!!!

Sync/Linking/mapping of JavaClasses with DB is Called OR-Mapping.

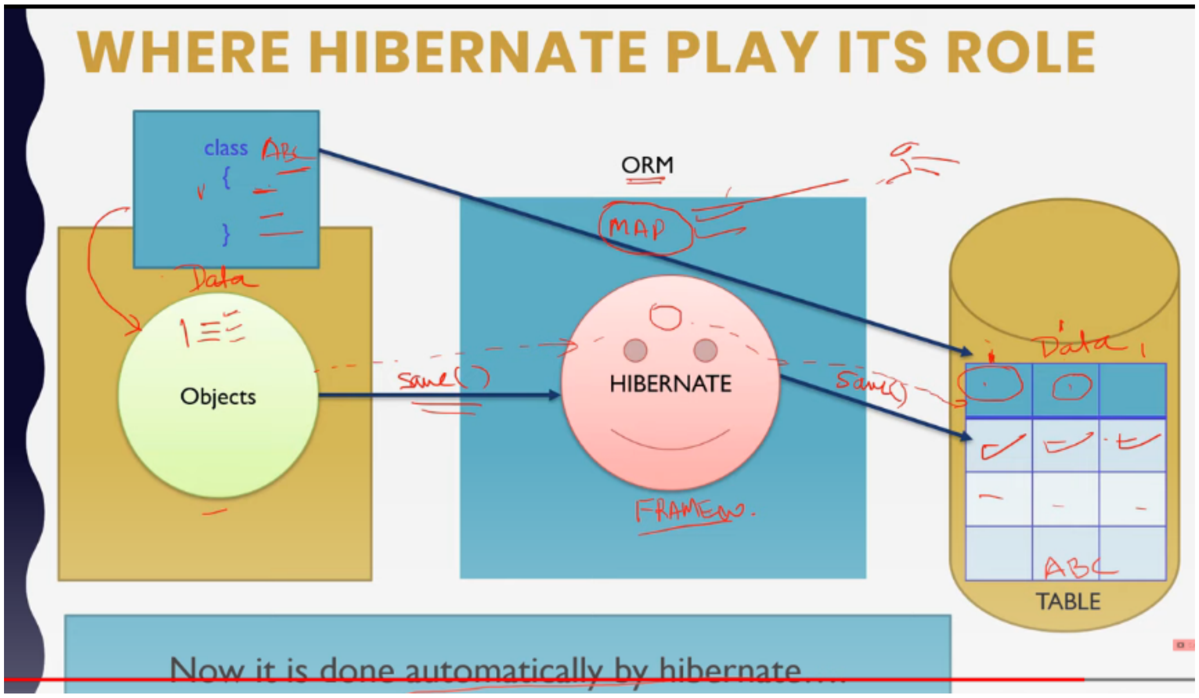
Spring and Hibernate Both are non-Invasive Frameworks i.e. are loosely Coupled and can be moved to different tech stack easily.

What is JPA -> Java Persistence API/Jakarta Persis?

The JPA is a set of rules/Specifications that every JPA provider like Hibernate or TopLink must follow to interact with DB. It Simplifies, Standardized the process

What is hibernate Framework ?

Java Framework Simplifies the working with db ORM tool Open Source , light Weight Non-invasive framework (wont't force programmer to implement any class/interface)


WHERE HIBERNATE PLAY ITS ROLE
class ABC
ORM
MAP
Data
Objects
save()
HIBERNATE
FRAMEW.
TABLE
ABC
Now it is done automatically by hibernate....

DAO Pattern of Hibernate

You need to create **Student class** (but you need to **map** it to the table)-> **DAO interface** -> DAO **implmenation** (Now Here all Operations would be without SQL) -> **Driver Manager(Main)**

now Mapping StudentClass to DB Table can be done with two ways **XML** Config or **java Annotations**

Maven Project Using Hibernate

```
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
<version>6.4.9.Final</version>
    <type>pom</type>
</dependency>
    <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
    <dependency>
      <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
     <version>8.0.33</version>
    </dependency>
```

Configuring Hibernate

there are two ways : hibernate.cfg.xml and hibernate.properties
Hibernate.cfg.xml (XML Based)

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration SYSTEM
"



http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd
">

<hibernate-configuration>
  <session-factory>

    // Things essential for database connectivity
```

```
        <p n="connection.driver.class"> com.mysql.cj.jdbc.Driver
         <p n="connection.url"> url
        <p n="connection.username"> root
        <p n="connection.password">  password
    //Dialect i.e.
         <p n="dialect" > org.hibernate.dialect.MySQLDialect
          // This will create a new table each time

          <p n="hbm2ddl.auto"> update
```

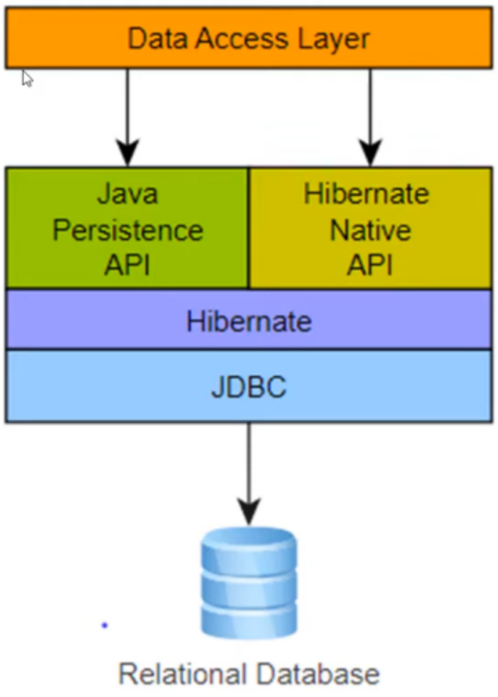
Dialect *one tells the hibernate which how to change query language withrespect to the provided database.*
The **hbm2ddL.auto** *tells the hibernate on how to ensure the schema of the table before applying any operation:*
These Commands are related to sessionFactory , it means changes are applied when sessionFactory is created and closed.
validate: *only validates the object mappings with table*
update: *updates the existing table according to the provided mapping*
create: *drops the current table and udpates the table according to the mapping provided*
create-drops: *same as the create drops the existing table and removes all the schema after the session factory is closed , so all the data is deleted.*
show_sql *: displays what queries are being used by hibernate behind the scenes on your console*

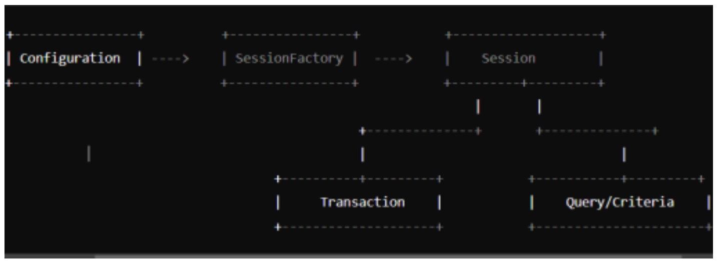
hibernate.properites (NON XML based)

The `hibernate.properties` file is a simpler, property-based configuration file. It serves the same purpose as the `hibernate.cfg.xml`
file but uses a different format.

```
# Database connection settings

hibernate.connection.driver_class=com.mysql.cj.jdbc.Driver
hibernate.connection.url=jdbc:mysql://localhost:3306/your_database
hibernate.connection.username=your_username
hibernate.connection.password=your_password

# JDBC connection poolsettings
hibernate.c3p0.min_size=5
hibernate.c3p0.max_size=20
hibernate.c3p0.timeout=300
hibernate.c3p0.max_statements=50
hibernate.c3p0.idle_test_period=3000

# SQL dialect
hibernate.dialect=org.hibernate.dialect.MySQLDialect

# Enable Hibernate's automatic session context management
hibernate.current_session_context_class=thread

# Echo all executed SQL to stdout
hibernate.show_sql=true

# Drop and re-create the database schema on startup
hibernate.hbm2ddl.auto=update
```

How to USE Hibernate / Architecture?





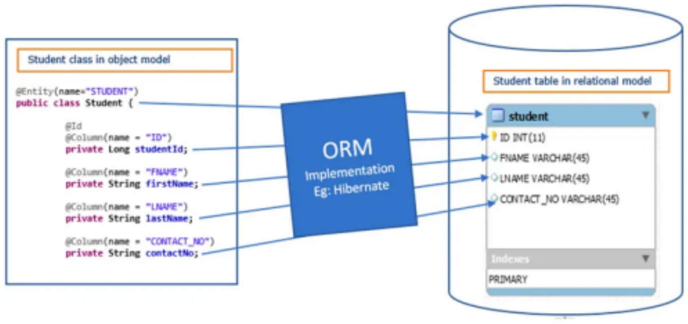
```
+---------------+       +---------------+       +---------------+
| Configuration |  ---->  | SessionFactory |  ---->  |    Session    |
+---------------+       +---------------+       +-------+-------+
                                                    |       |
                                          +---------+       +-----------+
        |                                 |                             |
                                  +-------+-------+             +-------+--------+
                                  |  Transaction  |             | Query/Criteria |
                                  +---------------+             +----------------+
```

Boilerplate Code

Driver Class Code :

```
Configuration config = new Configuration();
config.configure("hibernate.cnf.xml");
SessionFactory fact = config.buildSessionFactory();
```

Configuration - is used to read xml / to setup about database configuration ,
SessionFactory - used to create Session Objects to interact with Data Base



```
@Entity(name="STUDENT")
public class Student {

        @Id
        @Column(name = "ID")
        private Long studentId;

        @Column(name = "FNAME")
        private String firstName;

        @Column(name = "LNAME")
        private String lastName;

        @Column(name = "CONTACT_NO")
        private String contactNo;
```

Session - lightweight object that wraps around JDBC connection , responsible for Persistent CRUD operations

Transaction - used for data integrity , maintainbilty , these are not mandatory , but are recommended.

HQL query/criteria API : used for advanced and complex queries.

```
Session ses = factory.openSesssion();            ses.getTransaction();
ses.save(obj);
ses.getTransaction().commit();
ses.close();
```

Why use Transactions?

all operations between `beginTransaction()` and `commit()` are treated as a single unit. If any operation fails, none of the changes will be considered, and you can roll back the transaction to ensure the database remains consistent.

Entity/Object To be Mapped In Table Class :

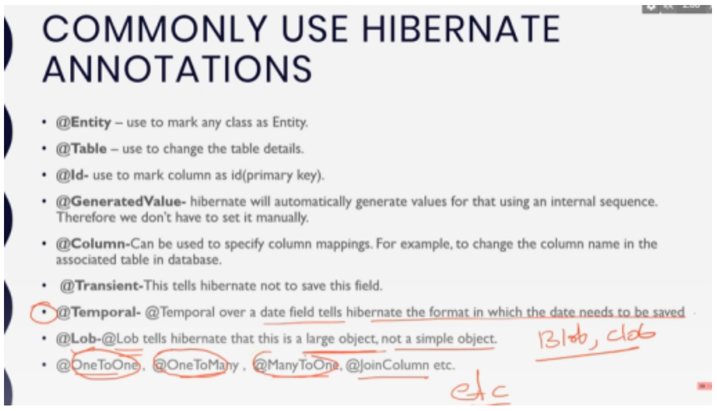
```
@Entity
@Table( name ="Table_Name")
public class Student{
   @Id
   fields;
   getter/Setter()
   constuctors;
 }
```

Note : Remember that u need to create all the getters and setters and all the default constructorss for the provided fields or else hibernate would not work properly.

Hibernate Config.xml :

```
<mapping name="com.Student" />
```

Hibernate Annotations



COMMONLY USE HIBERNATE ANNOTATIONS

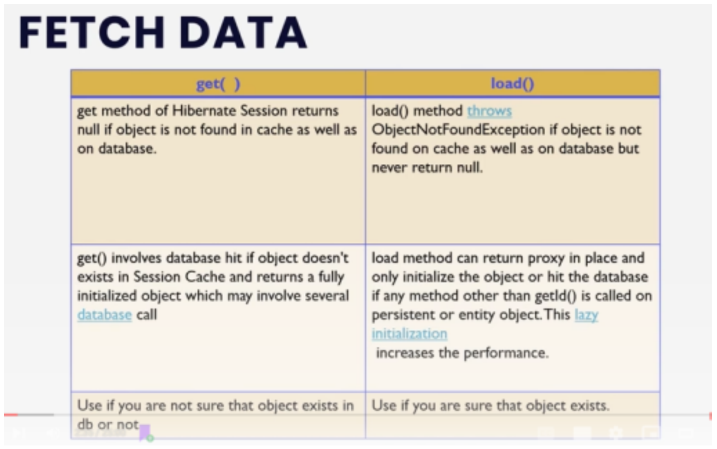
- **@Entity** – use to mark any class as Entity.
- **@Table** – use to change the table details.
- **@Id**- use to mark column as id(primary key).
- **@GeneratedValue**- hibernate will automatically generate values for that using an internal sequence. Therefore we don't have to set it manually.
- **@Column**-Can be used to specify column mappings. For example, to change the column name in the associated table in database.
- **@Transient**-This tells hibernate not to save this field.
- **@Temporal**- @Temporal over a date field tells hibernate the format in which the date needs to be saved
- **@Lob**-@Lob tells hibernate that this is a large object, not a simple object. Blob, clob
- @OneToOne, @OneToMany , @ManyToOne, @JoinColumn etc.

etc

How to store image in DB ?

```
FileInputStream file = new FileInputStream("path");
byte[] img = new byte[file.available()];
```

How to get data from DB / Select Command



FETCH DATA

get( )	load()
get method of Hibernate Session returns null if object is not found in cache as well as on database.	load() method throws ObjectNotFoundException if object is not found on cache as well as on database but never return null.
get() involves database 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 method other than getId() is called on persistent or entity object. This lazy initialization increases the performance.
Use if you are not sure that object exists in db or not.	Use if you are sure that object exists.

`session.get()` when you need the actual object immediately and want to handle null values and also it fires the query once and stores it into session cache, so what happens is , **if u again ask for that object , it will not fire query to db it will first check its cache**. **`session.load()`** when you are sure the entity exists and are okay with a proxy object.It returns a placeholder not the actual object,**it means the query is not fired to the DB until u have not accessed a method or field of that object**.

@Embeddable -

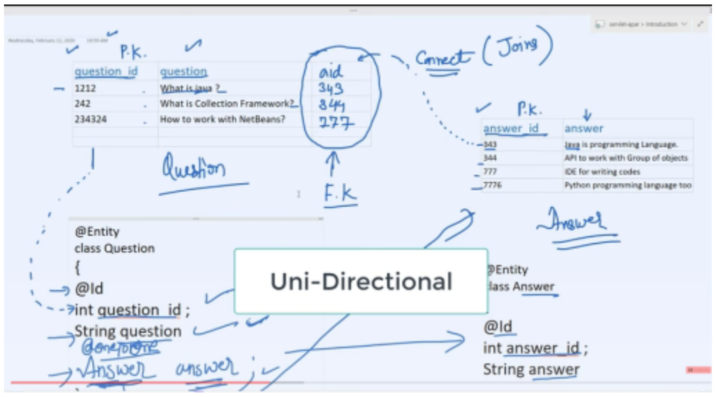
In summary, the `@Embeddable` annotation in Hibernate allows you to encapsulate multiple related fields into a single class, which can then be embedded in multiple entities. This promotes modularity, simplifies entity definitions.

```
@Entity
public class Customer {
   @Id
     private int id;
     private String name;

   @Embedded
     private Address address;
```

}
now this entity class talble in DB will contain all the feilds of the address class, like here it is city , state etc. in its own table.

Mappings

1) One to One



question_id	question	aid
1212	What is java ?	343
242	What is Collection Framework?	344
234324	How to work with NetBeans?	777

answer_id	answer
343	Java is programming Language.
344	API to work with Group of objects
777	IDE for writing codes
7776	Python programming language too

```
@Entity
class Question
{
@Id
int question_id ;
String question
@OneToOne
Answer answer ;
```

```
@Entity
class Answer

@Id
int answer_id ;
String answer
```

To create a **unidirectional mapping** just add
@OneToOne to the field that is going to be mapped

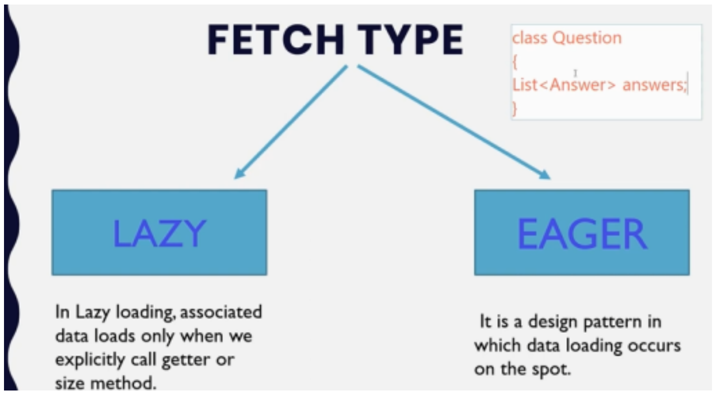
And for **Bidirectional** u need to use **@OneToOne** on the one side and on the other Entity u have to use **@OneToOne(mappedBy = "FirstEntityMappedField")**
This mappedBy prevents from creating two **duplicate foreign** keys for each table.. , now there should be only one foreign key that would be present in the first Entity.

For this OneToOne mapping we only use a single object reference but for others we need to have a list of objects or collection.

we can say that **mappedBy indicates on which side of entity / where that "Variable_Name" is present will have the foreign key.**
So suppose ur using MappedBy in Entity A then Entity B will have the foreign Key.
And the entity which have the f**oreign key in its table own the relationship**

cascade = CascadeType.All or any other type basically tells hibernate that suppose if there are two entities related to each other like A and B and if u saved A then the hibernate will automaticlly will save B , and same goes for all other operations mentioned like CascadeType.ALL , REMOVE, SAVE,DELETE,DETACH etc..

fetch = FetchType.Lazy so this fetches / retrieves entities from DB when they are accessed / similar to how session.load() works
fetch = FetchType.Eager so this fetches/retrives data loading occurs at the spot



```
class Question
{
List<Answer> answers;
}
```

In Lazy loading, associated data loads only when we explicitly call getter or size method.

It is a design pattern in which data loading occurs on the spot.

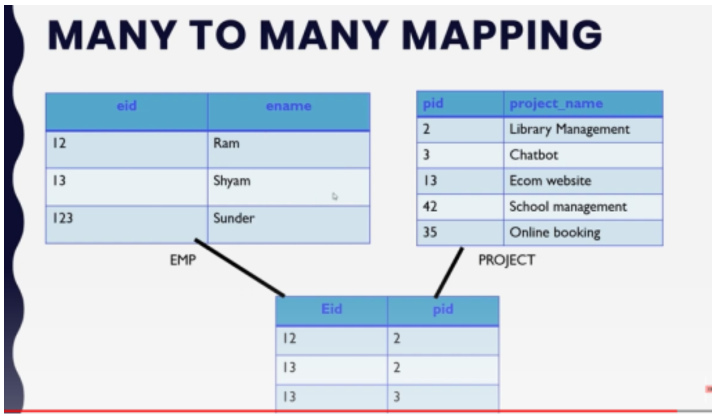
2) OneToMany

The entity which will be mapped with like
student can opt courses , or questions can have mulitple answers then

```
Questions{

   @OneToMany( mappedBy = "question")
    List<Answer> answer;
    // Remember it should be a collection
}
                                                   Answer{

   @ManyToOne
    Question question;
}
```

Note : If u will not use mappedBy attribute then there will be a third table created denoting the foreign keys which is not good , thats why we use mappedBy to tell hibernate that the foreign keys of this table are mapped by the other attribute of that entity.

3) ManyToMany



MANY TO MANY MAPPING

EMP

eid	ename
12	Ram
13	Shyam
123	Sunder

PROJECT

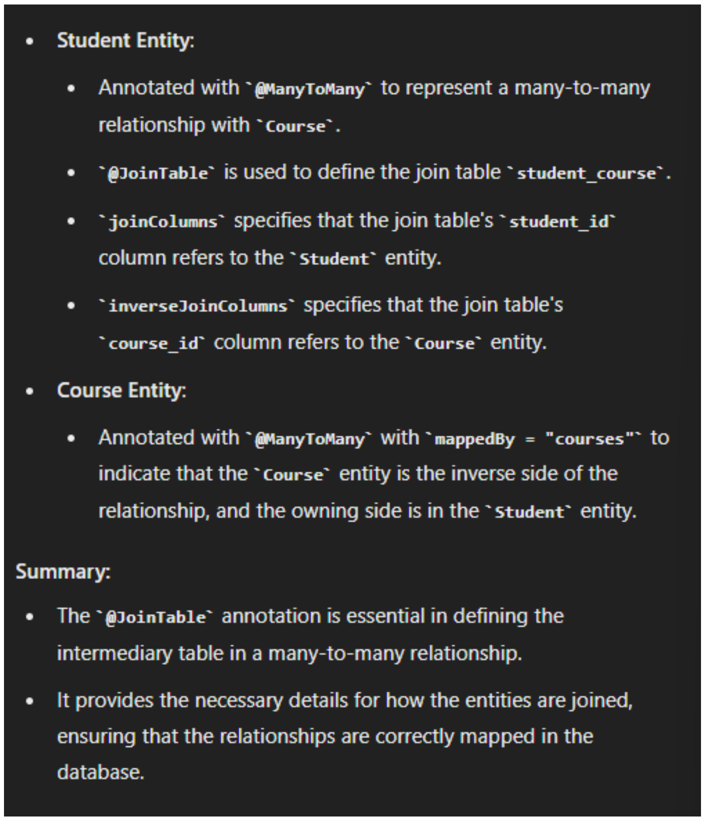
pid	project_name
2	Library Management
3	Chatbot
13	Ecom website
42	School management
35	Online booking

Eid	pid
12	2
13	2
13	3

Here Both entities will have @ManyToMany Annotation and if MappedBy attribute is not used then there will be two tables created for Refrential Intigrity.
Note : First Create and Save all the entities in the DB and then form / create any relation between them , otherwise some errors from hibernate may occur.

@JoinTable()

```
@ManyToMany
@JoinTable(
     name = "Join_table",
     joinColumns = @JoinColumn(name="Entity_A"),
    inverseJoinColumns = @JoinColumn(name="E_B")
)
```


- **Student Entity:**
  - Annotated with `@ManyToMany` to represent a many-to-many relationship with `Course`.
  - `@JoinTable` is used to define the join table `student_course`.
  - `joinColumns` specifies that the join table's `student_id` column refers to the `Student` entity.
  - `inverseJoinColumns` specifies that the join table's `course_id` column refers to the `Course` entity.
- **Course Entity:**
  - Annotated with `@ManyToMany` with `mappedBy = "courses"` to indicate that the `Course` entity is the inverse side of the relationship, and the owning side is in the `Student` entity.

Summary:

- The `@JoinTable` annotation is essential in defining the intermediary table in a many-to-many relationship.
- It provides the necessary details for how the entities are joined, ensuring that the relationships are correctly mapped in the database.

It provides option for Explicit Table Name and Explicit Joining Column Names unlike mappedBy which uses Default

Note : If some error occurs , of u wanna delete the table then do it in this manner :

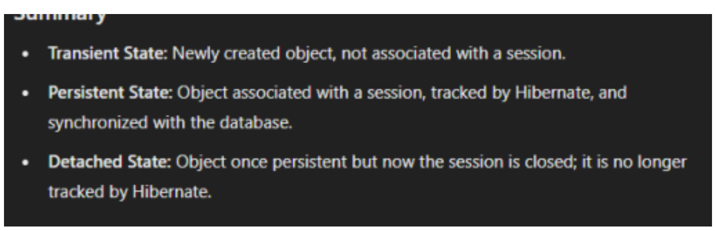
1) Drop Foreign Key Constraints

ALTER TABLE project_employe DROP FOREIGN KEY FKa93bvg59wttv3el9pi9iei746;ALTER TABLE project_employe DROP FOREIGN KEY FK1fgidoat22wfb443wl29oswwi;

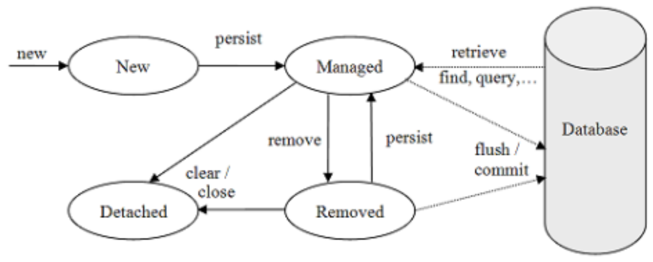
2) Drop the table

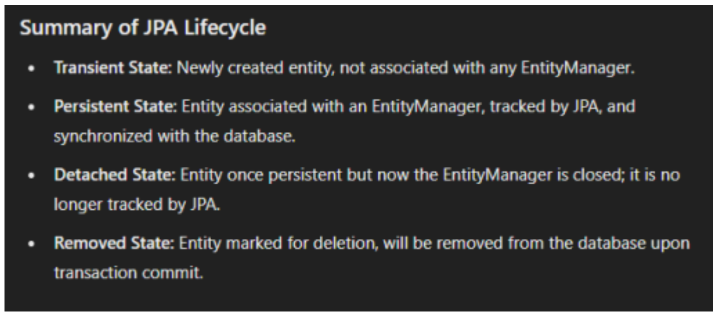
DROP TABLE IF EXISTS project_employe;
DROP TABLE IF EXISTS project;
DROP TABLE IF EXISTS employe;

Hibernate Object / Persistent LifeCycle


- **Transient State:** Newly created object, not associated with a session.
- **Persistent State:** Object associated with a session, tracked by Hibernate, and synchronized with the database.
- **Detached State:** Object once persistent but now the session is closed; it is no longer tracked by Hibernate.

It is the same as JPA LifeCycle just that there is an extra state , which shows the removal of object from DB.





Summary of JPA Lifecycle

- **Transient State:** Newly created entity, not associated with any EntityManager.
- **Persistent State:** Entity associated with an EntityManager, tracked by JPA, and synchronized with the database.
- **Detached State:** Entity once persistent but now the EntityManager is closed; it is no longer tracked by JPA.
- **Removed State:** Entity marked for deletion, will be removed from the database upon transaction commit.