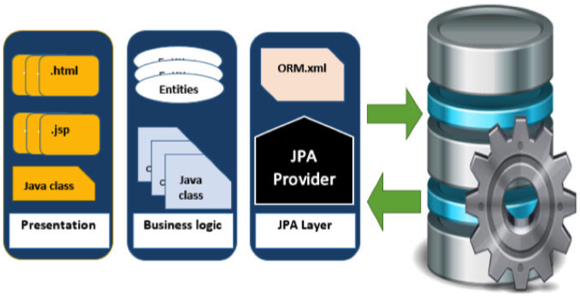
Query creation

https://docs.spring.io/spring-data/jpa/docs/current/reference/html/

<https://www.tutorialspoint.com/jpa/index.htm>

Java Persistence API is a collection of classes and methods to persistently store the vast amounts of data into a database. (persistence: kiên trì, như kiểu quan hệ ràng buộc relational)

It forms a bridge between object models (Java program) and relational models (database program).



Java Persistence API is a source to store business entities as relational entities. It shows how to define a PLAIN OLD JAVA OBJECT (POJO) as an entity and how to manage entities with relations.

## **Class Level Architecture**

## Đọc trong link:

## <https://www.tutorialspoint.com/jpa/jpa_architecture.htm>

# **JPA - ORM Components**

The core part of this object relational technologies are mapping orm.xml file.

The main feature of ORM is mapping or binding an object to its data in the database. While mapping we have to consider the data, type of data and its relations with its self-entity or entity in any other table.

<https://www.tutorialspoint.com/jpa/jpa_orm_components.htm>

# **JPA - Entity Managers**

The main modules for this example are as follows:

* **Model or POJO**

Employee.java

* **Persistence**

Persistence.xml

* **Service**

CreatingEmployee.java

UpdatingEmployee.java

FindingEmployee.java

DeletingEmployee.java

<https://www.tutorialspoint.com/jpa/jpa_entity_managers.htm>

# **JPA - JPQL**

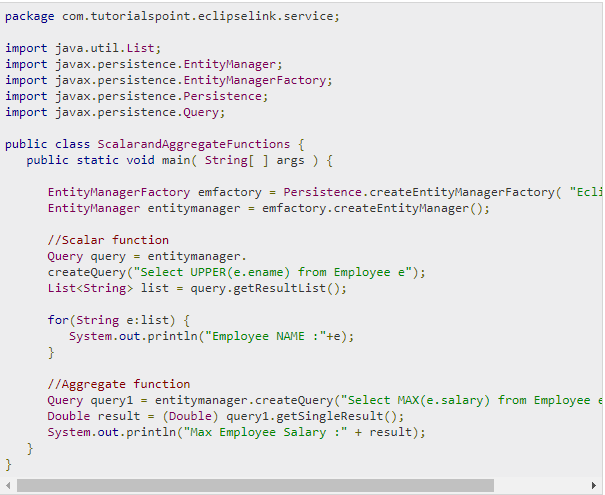
JPQL is Java Persistence Query Language defined in JPA specification. It is used to create queries against entities to store in a relational database. JPQL is developed based on SQL syntax. But it won’t affect the database directly.

EntityManager.createQuery() API will support for querying language.

JPQL syntax is very similar to the syntax of SQL.

## **Scalar and Aggregate Functions**

Scalar functions returns resultant values based on input values. Aggregate functions returns the resultant values by calculating the input values.



## **Named Queries** (https://www.tutorialspoint.com/jpa/jpa\_jpql.htm)

## **Eager and Lazy Loading**

The main concept of JPA is to make a duplicate copy of the database in cache memory. While transacting with the database, first it will effect on duplicate data and only when it is committed using entity manager, the changes are effected into the database.

There are two ways of fetching records from the database - eager fetch and lazy fetch.

### **Eager fetch**

Fetching the whole record while finding the record using Primary Key.

### **Lazy fetch**

It checks for the availability of notifies it with primary key if it exists. Then later if you call any of the getter method of that entity then it fetches the whole.

But lazy fetch is possible when you try to fetch the record for the first time. That way, a copy of the whole record is already stored in cache memory. Performance wise, **lazy fetch is preferable**.

(https://stackoverflow.com/questions/2990799/difference-between-fetchtype-lazy-and-eager-in-java-persistence-api

**students to be loaded when they are actually needed. This is called lazy loading.)**

Ví dụ: University entity có nhiều Student

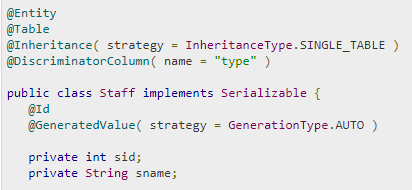
**Eager fetch: khi load university nó sẽ load toàn bộ cả 2 university và student**

**Lazy fetch: chỉ load university và load student nếu khi nào nó cần**

# **JPA - Advanced Mappings**

JPA support three types of inheritance strategies such as SINGLE\_TABLE, JOINED\_TABLE, and TABLE\_PER\_CONCRETE\_CLASS.

## **Single Table strategy**



## **Joined Table strategy**

@Inheritance( strategy = InheritanceType.JOINED )

## **Table per class strategy**

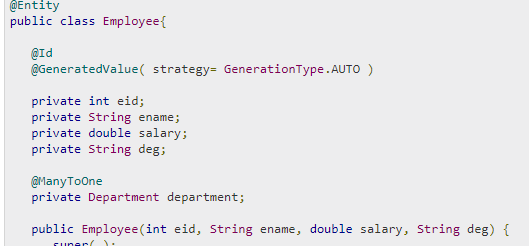
@Inheritance( strategy = InheritanceType.TABLE\_PER\_CLASS )

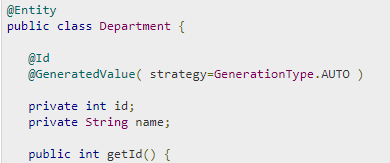
# **JPA - Entity Relationships**

* @ManyToOne Relation
* @OneToMany Relation
* @OneToOne Relation
* @ManyToMany Relation

**@ManyToOne:**

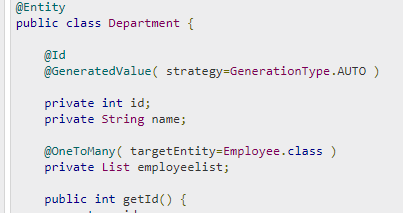
Ví dụ: nhiều Employee trong 1 căn hộ Department

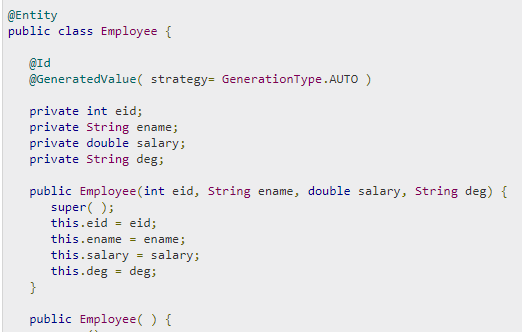




**@OneToMany : ngược lại với ManyToOne ở trên**

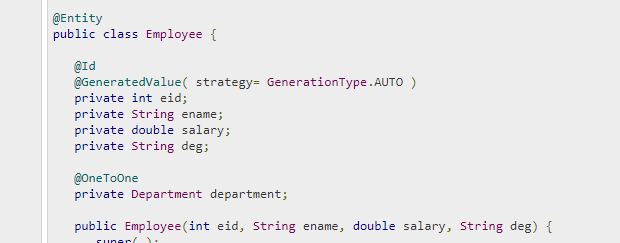
In this relationship each row of one entity is referenced to many child records in other entity.

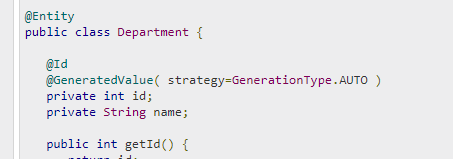




## @OneToOne

Ví dụ: 1 Department chỉ cho 1 Employee sống







**@ManyToMany**

Một lớp có nhiều giáo viên dạy, Một giáo viên dạy nhiều lớp

