Hibernate/JPA Crud

- Hibernate handles all low-level SQL.
- ORM: Object-To-Relational Mapping (ORM)
  Java annotations

$$\begin{tikzcd}\boxed{\text{class}} \arrow[r, dash] & \boxed{\text{Hibernate}} \arrow[r, dash] & \boxed{\text{SQL}}\end{tikzcd}$$

- JPA: Standard API for ORM. Hibernate is a JPA like Eclipselink is JPA
  entityManager:
- Retrieve Object with JPA: entityManager.find(Student.class, theId);
- Pas besoin faire low-level SQL
- Save Object with JPA: entityManager.persist(student);

Hibernate/JPA and JDBC

layer of JDBC

JDBC: Java Database Connectivity
↳ Allows Java Program to access database and other data resources.

spring student

des configurations db dans application.properties

CommandLineRunner → • executed when all beans are loaded
- run specific code at startup

✓✓

Entity class :
- @Entity
- public / protected constructor
- Java Class that is mapped to a database table.

1. Map Class to database table
2. Map fields to database columns

→ Different naming (optional) @Column(name = "field_in_db")
if not specified, column name same as Java field.

Same for @Table → for table name

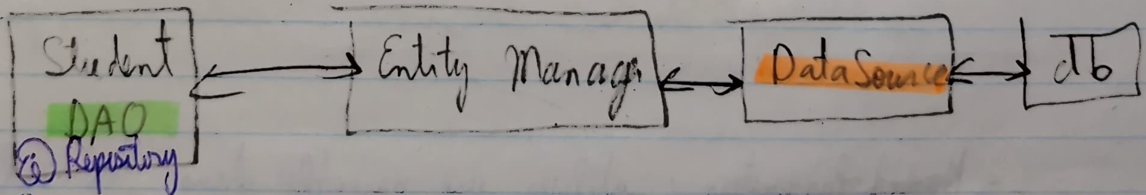
- PK Generated Values

<u>ID Generation Strategies</u>

Auto
Identity
Sequence
table
custom strategy

Entity Manager : main component creating queries

Data Access Object : responsable interact with db
(aka DAO)



→ Comme <u>helper class</u> qui communique avec db

Datasource : - connection info
- created by Spring boot
ex: save();

Jpa Repository vs Entity Manager

Jpa Repository:

- Use it when high level repository
- out of the box CRUD operations
- Custom query using @Query

Entity Manager:

- store procedures
- rare SQL

@Transactional

- Auto <u>begin</u> and <u>end</u> transaction for JPA code
- Mettre en-tête méthode → Add, update

@Repository

- for DAO annotations
- register DAO implementation

- Save: entityManager.persist( )
- Read: entityManager.find(Student.class, 1)
  ↳ returns null if don't find

- Query objects:
  - JPQL → retrieve objects
  - use things like where, like, join
  - Use entity name and entity fields, not table

TypedQuery<Student> theQuery = entityManager.createQuery

- Named Parameters: utiliser une variable dans ton entity manager

- findAll: entityManager.createQuery

Update : 1. find
2. perform the update → entityManager.merge(student);

delete : 1. find
2. remove()
ou bien
entityManager.createQuery("DELETE FROM Student")
.executeUpdate();

Create db from Java Code:
application.properties → spring.jpa.hibernate.ddl-auto=create

Many types like:
- create : every time run : drop table, create table
- update : keep previous data