

TP2 – INTERFACAGE DE BASES DE DONNÉES

AVEC HIBERNATE, NOSQL

Par  
Enrik Bernier et Ludovic Hébert

Travail présenté au professeur Mehdi Adda

Dans le cadre du cours Base de données IIINF25207-MS

15 décembre 2024

Table des matières

[Associations 2](#_Toc185042710)

[Exemples d’association 2](#_Toc185042711)

[Un à un 2](#_Toc185042712)

[Un à plusieurs 4](#_Toc185042713)

[Plusieurs à plusieurs 6](#_Toc185042714)

[Choix de conceptions 8](#_Toc185042715)

[Références pour l’étape d’association 8](#_Toc185042716)

[Validation de données 9](#_Toc185042717)

[Processus création validateur 9](#_Toc185042718)

[Références pour l’étape de validation 9](#_Toc185042719)

[Partie Bonus 9](#_Toc185042720)

[Hibernate Search 9](#_Toc185042721)

[Références partie bonus 9](#_Toc185042722)

[Répartition des tâches 9](#_Toc185042723)

# Associations

## Exemples d’association

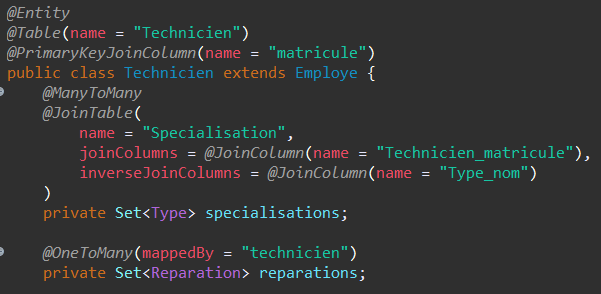
### Un à un

Pour les relations un à un, nous avons continué à utiliser la méthode montrée en cours, en donnant la charge de la possession de la clé étrangère (mapping) à la classe qui possédait la clé étrangère dans le diagramme. Dans ce cas-ci, les tables Pilote et Technicien possédaient la relation avec la table employée. Puisqu’il s’agit ici d’un cas très clair d’héritance, la classe d’origine Employe utilise la propriété @Inheritance(strategy = InheritanceType.JOINED) pour indiquer à Hibernate que Technicien et Pilote doivent être séparé de Employe et ses attributs et simplement avoir une référence vers l’employé correspondant.



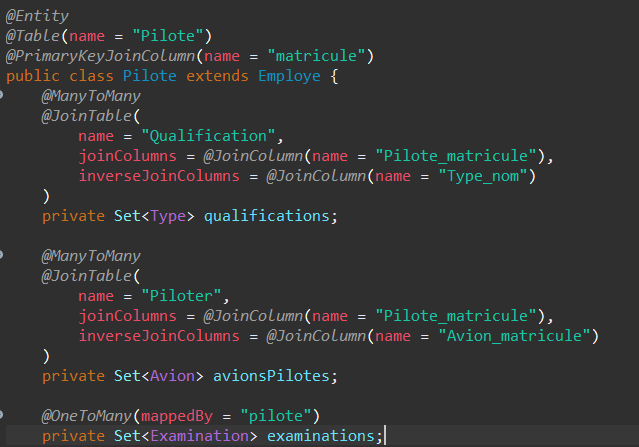
*Image 1 : Classe Employe et ses annotations*

1.Employé – Technicien :

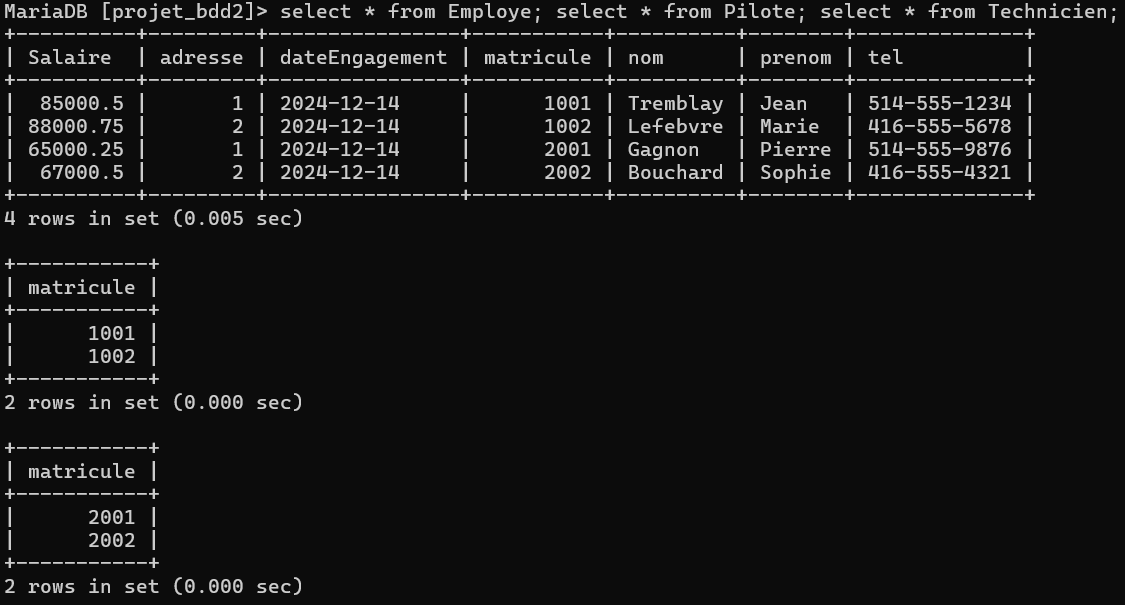


*Image 3 : Classe Technicien et ses annotations*

2.Employe – Pilote :



*Image 4 : Classe Pilote et ses annotations*



*Image 5 : Tables résultantes de la configuration Employé – Technicien et* *Employé – Pilote*

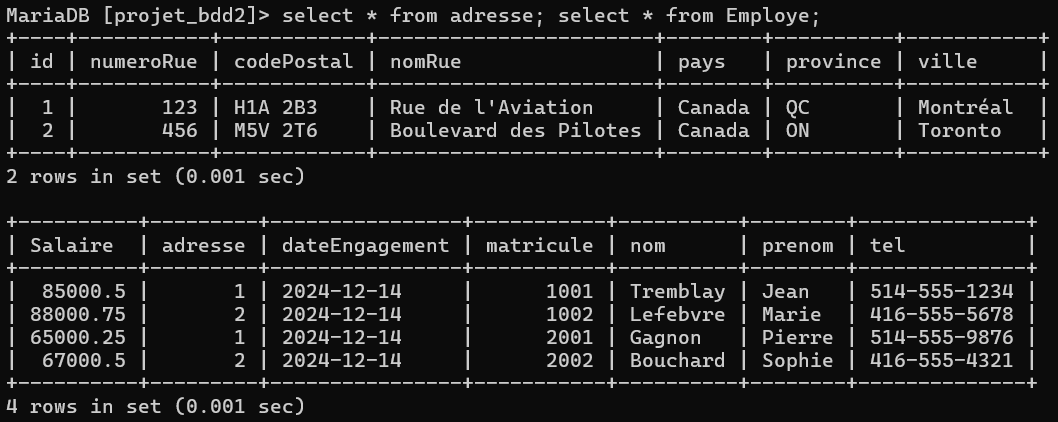
### Un à plusieurs

Pour les relations un à plusieurs, nous avons utilisé la même méthode qu’en cours, en donnant à la table du côté multiple de la relation de stocker les clés étrangères de la table du côté unaire. Pour la relation Adresse – Employe, c’était donc Employé, et pour Type – Avion, Avion, puisque plusieurs employés peuvent avoir une même adresse et chaque employé a une adresse, et un avion peut faire partie d’un seul type et plusieurs avions peuvent avoir le même type.

1.Adresse – Employe :

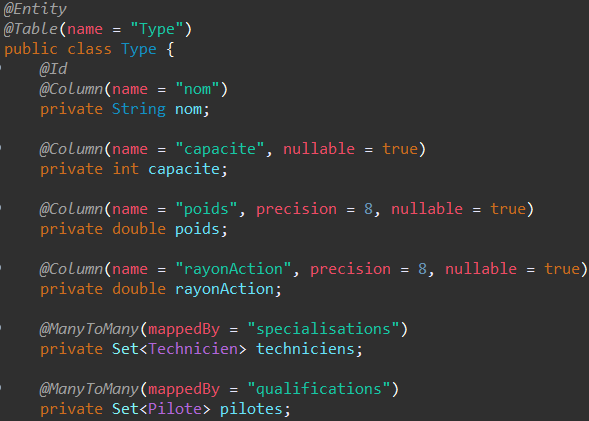


*Image 6 : Classe Adresse et ses annotations*



*Image 7 : Tables résultantes de la configuration Adresse – Employé*

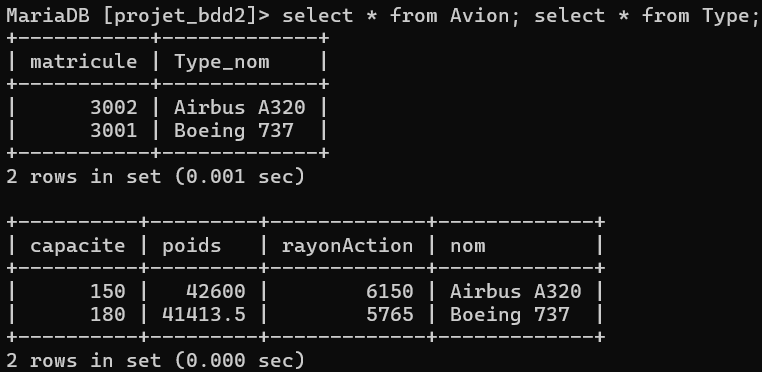
2.Type - Avion

**

*Image 8 : Classe Type et ses annotations*

**

*Image 9 : Classe Avion et ses annotations*

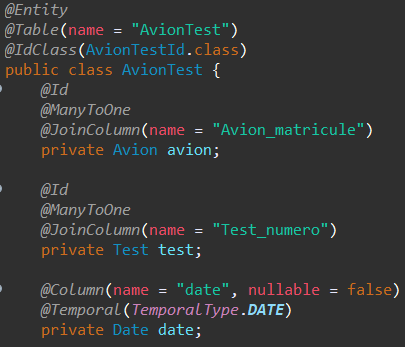


*Image 10 : Tables résultantes de la configuration Type – Avion*

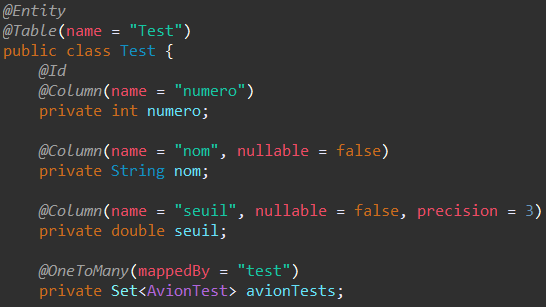
### Plusieurs à plusieurs

Finalement, pour les relations plusieurs, la même chose s’applique, à quelques différences près : certaines tables, comme AvionTest, sont des tables pour des relations plusieurs à plusieurs. Elles n’ont de sens qu’avec les deux tables qu’elles référencent, mais elles possèdent une ou des colonnes contenant de l’information supplémentaire sur la relation entre deux tuples des tables (comme date avec AvionTest). Dans ces cas, nous avons créé une classe à part pour ces tables avec des relations un à plusieurs avec les deux autres tables (Avion et Test pour AvionTest). Pour les tables plus simples comme Piloter, nous avons déclaré la table de jointure dans les classes des deux tables (Pilote et Avion pour Piloter).

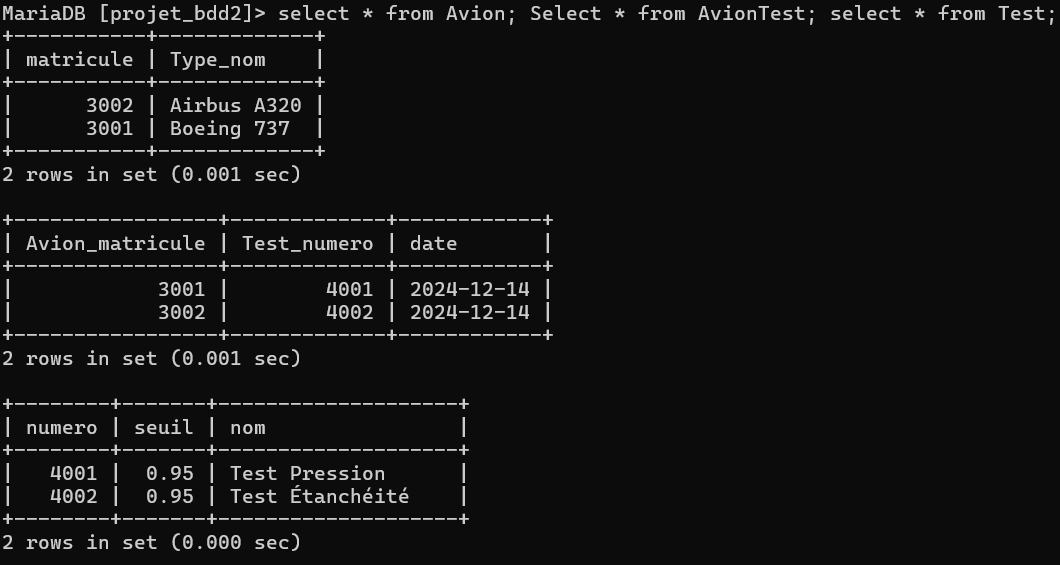
1.AvionTest



*Image 11 : Classe AvionTest et ses annotations*



*Image 12 : Classe Test et ses annotations*

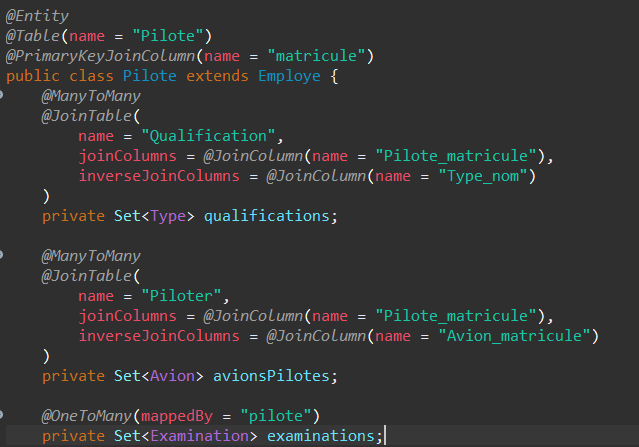


*Image 12 : Tables résultantes de la configuration Avion – AvionTest – Test*

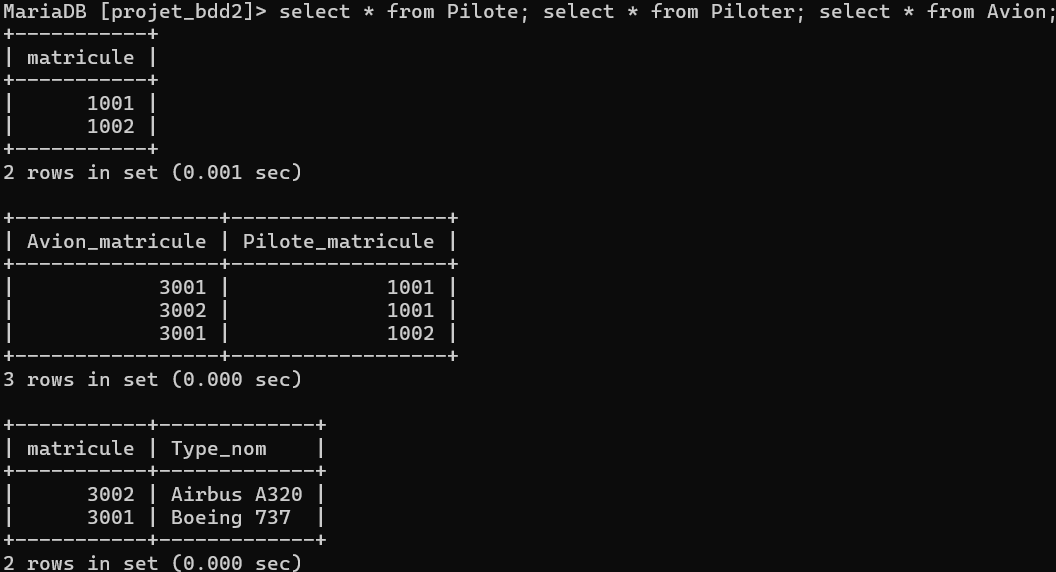
2.Piloter

**

*Image 13 : Classe Avion et ses annotations*



*Image 14 : Classe Pilote et ses annotations*

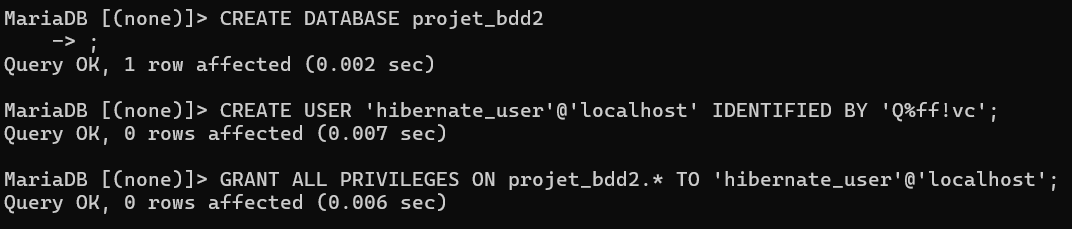


*Image 15 : Tables résultantes de la configuration Avion – Piloter – Pilote*

## Choix de conceptions

Nous avons choisi de faire notre projet un projet Maven, puisque lors du début du projet nous éprouvions beaucoup de difficultés à faire fonctionner notre projet à cause des dépendances manquantes. Le passage à Maven à permis d’alléger ce fardeau. Nous avons suivi l’exemple de configuration du projet dans le portail UQAR pour la configuration de projets Maven avec Hibernate.

Le pilote et la base de données sont tous les deux pour Maria dB, puisque c’est la fourche de MySQL avec laquelle nous sommes la plus familière. Pour les versions des dépendances utilisées, nous avons généralement utilisé les plus récentes puisque nous utilisions JDK 23.



*Image 16 : Création de la base de données et de l’utiilisateur*

Pour les clés composées comme dans AvionTest, nous avons utilisé IdClass pour faciliter la gestion de celle-ci.

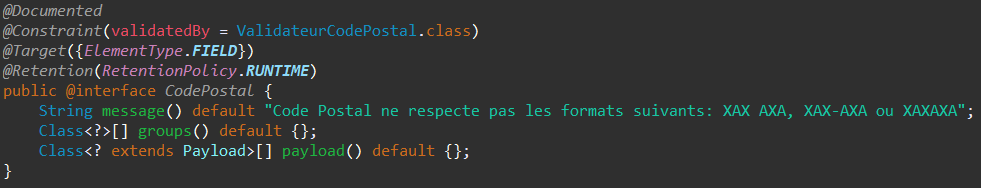
## Références pour l’étape d’association

* Vivek Balasubramaniam, Baeldung.com, “Composite Primary Keys in JPA”, 11 Mai 2024, <https://www.baeldung.com/jpa-composite-primary-keys>

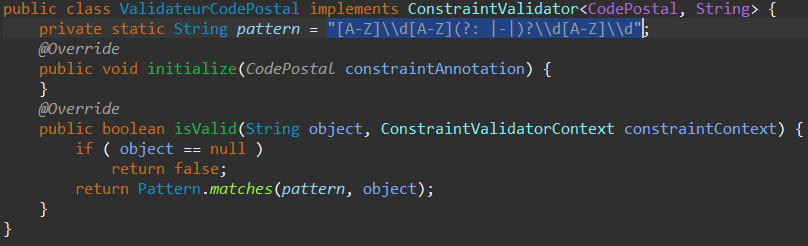
# Validation de données

## Processus création validateur

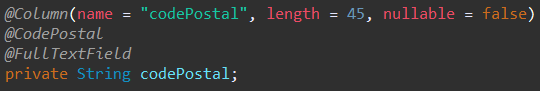
Pour la création du validateur, nous avons utilisé la documentation officielle pour hibernate validator. Cette étape s’est passée sans réel blocage, à part le temps pour trouver comment faire un regex avec une partie optionnelle (espace, tiret ou rien). Une simple recherche sur google a permis de trouver une source qui montrait comment faire.



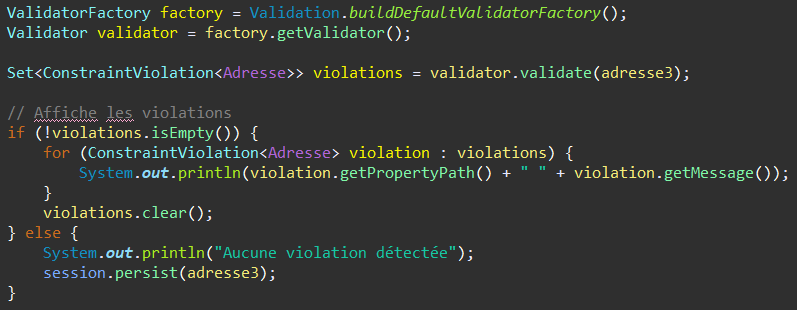
*Image 17 : Création de l’annotation CodePostal*



*Image 18 : Création du validateur utilisé par l’annotation CodePostal*



*Image 19 : Ajout de l’annotation CodePostal pour la vérification de la validité de l’attribut*



*Image 20 : Création du validateur et vérification d’un code postal*

## Références pour l’étape de validation

* Hardy Ferentschik, Gunnar Morling, Guillaume Smet. “Hibernate Validator 8.0.2. Final - Jakarta Bean Validation Reference Implementation” 13 Décembre 2024, <https://docs.jboss.org/hibernate/stable/validator/reference/en-US/pdf/hibernate_validator_reference.pdf>
* Oracle Help Center, “Class pattern” <https://docs.oracle.com/javase/6/docs/api/java/util/regex/Pattern.html>

# Partie Bonus

## Hibernate Search

Pour Hibernate search, nous avons utilisé le tutoriel fournit sur leur site web pour la partie d’interrogation des données. Toutefois il n’était pas indiqué comment obtenir le EntityManager utilisé, nous avons donc dû chercher ailleurs comment l’obtenir. Nous avons trouvé dans SessionFactory la méthode createEntityManager() et l’avons utilisé pour la suite.

À part cela, nous n’avons eu des problèmes que pour la configuration initiale de Hibernate search, puisqu’ils utilisaient Hibernate bom pour aligner les versions de leurs dépendances, mais cette fonctionnalité n’était pas compatible avec notre fichier pom. Nous avons donc dû manuellement retrouver les numéros de versions. Finalement, la dépendance jboss.logging n’était pas indiquée, mais nécessaire pour l’exécution.

## Références partie bonus

* Hibernate, “Getting started with Hibernate Search in Hibernate ORM”, 25 novembre 2024, <https://docs.jboss.org/hibernate/search/7.2/getting-started/orm/en-US/html_single/index.html>

# Répartition des tâches

Ludovic Hébert a commencé le travail en préparant un GitHub. Ensuite, nous avons travaillé sur la configuration des classes en s’assurant que les contraintes soient respectées et que les entités définies soient fidèles au diagramme présenté dans l’énoncé de travail. Ludovic a pris l’initiative sur la configuration de Hibernate pour la vérification des données ainsi que pour la librairie Hibernate Search. Ludovic a préparé le rapport et Enrik a ajouté des modifications pour améliorer la présentation du rapport.

Enrik Bernier:

* Configuration des classes et annotations
* Corrections au rapport

Ludovic Hébert:

* Configuration des classes et annotations
* Configuration de hibernate verification et search
* Préparation du rapport