

# La spécification JPA 2.1

## Les entités

30/09/2017

Walid YAICH

[walid.yaich@esprit.tn](mailto:walid.yaich@esprit.tn)

Bureau E204



# Plan

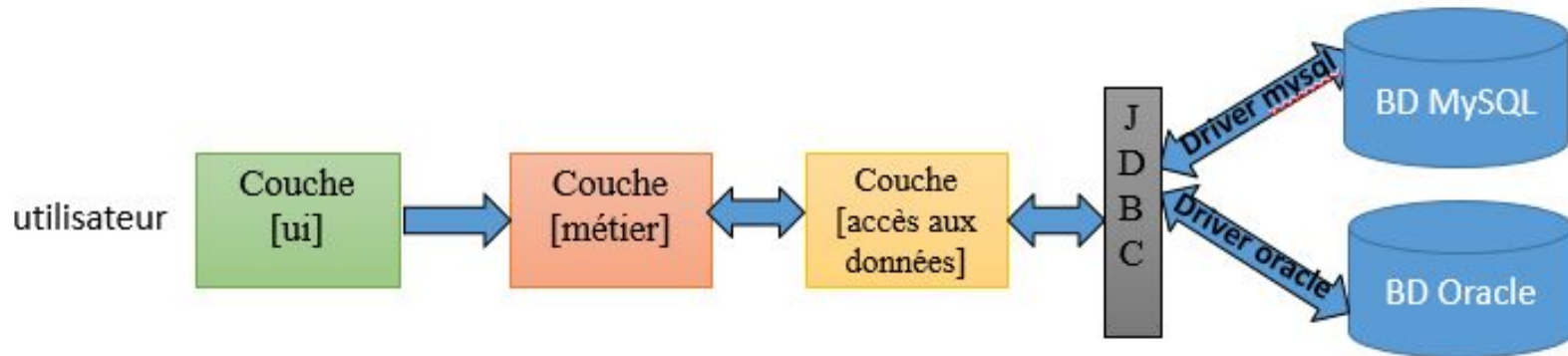
- JDBC
- ORM
- Introduction JPA
- La sérialisation
- Les entités JPA
- Basic Types
- Le type `java.util.Date`
- Stratégies d'auto génération des IDs
- Les énumérations
- Classe Embeddable
- Clé composite
- Clé composite avec `@IdClass`
- Clé composite avec `@EmbeddedId`
- Héritage
- Héritage single Table (par défaut)
- Persistence.xml
- Création de la premiere entité

# JDBC

L'utilisation de l'API JDBC était une solution pour les développeurs java pour unifier l'accès à une base de données SQL.

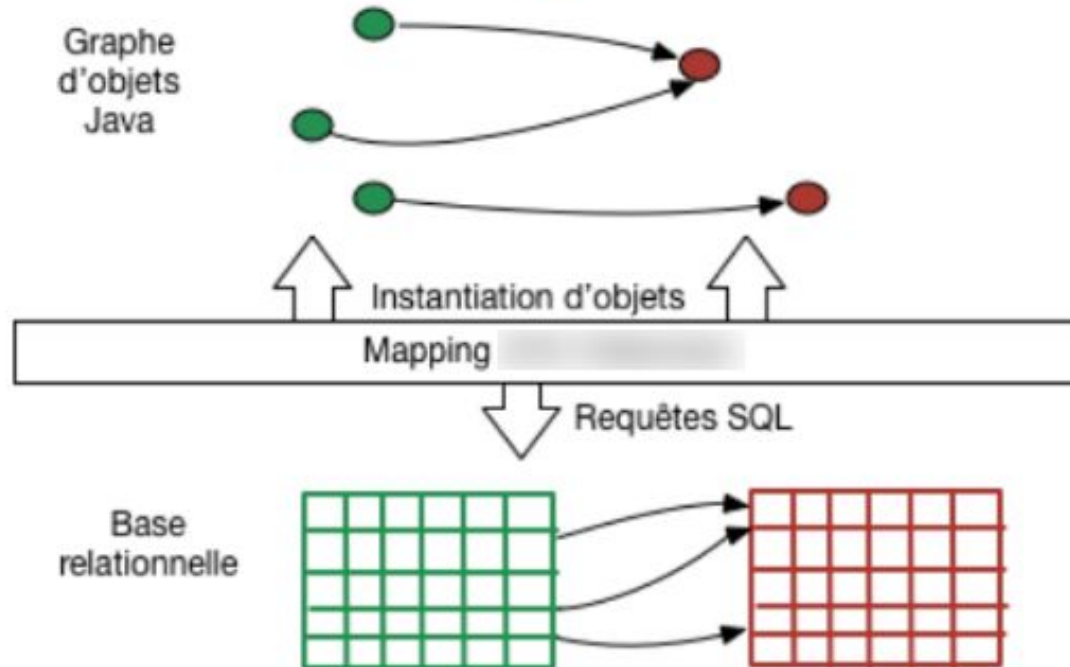
Inconvénients :

- Pas de séparation entre le code technique et le code métier
- utilisation du langage SQL rend la couche d'accès aux données **difficilement maintenable**.



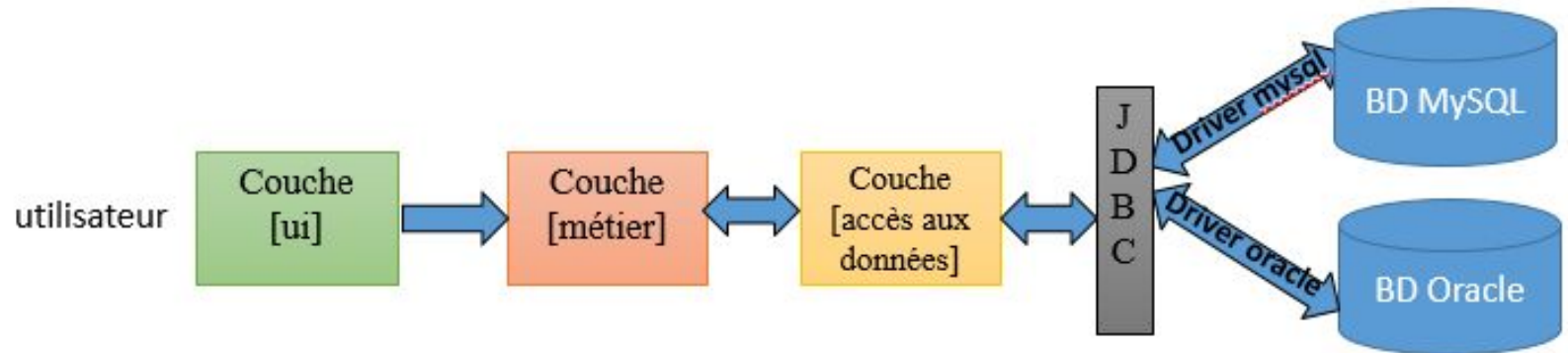
# ORM

L'ORM (Object-Relational Mapping) est un concept faisant le lien entre le monde de la base de données et le monde de la programmation orienté objet.

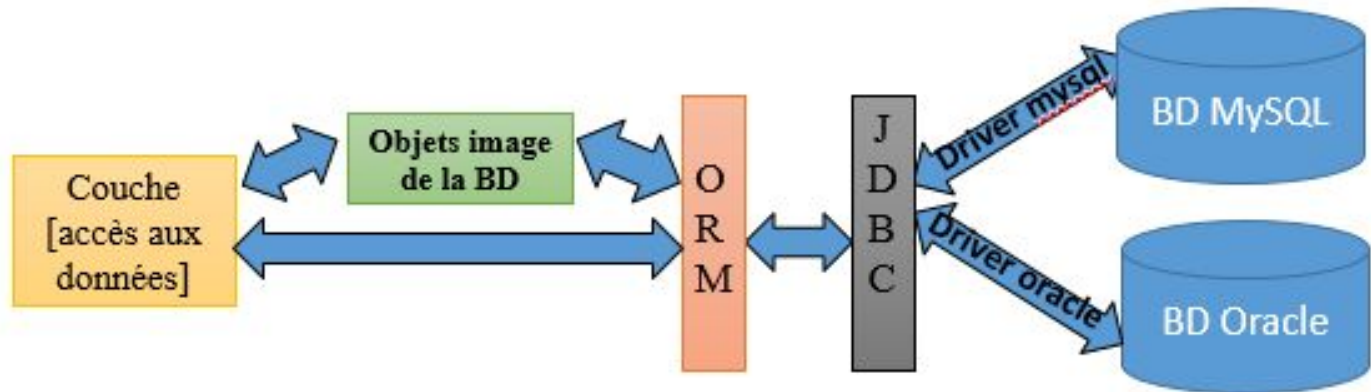


# ORM

Sans  
ORM



Avec  
ORM



# Introduction JPA

JPA est une **spécification** Java EE basée sur le concept ORM (Object Relational mapping).

L'utilisation de JPA nécessite un **fournisseur de persistance** qui implémente les spécifications JPA.

Il y a plusieurs fournisseurs de persistance qui implémentent la spécification JPA, tel que, **Hibernate, Toplink, datanucleus...**

Dans ce cours nous allons utiliser **Hibernate** comme implémentation de la spécification JPA.

# Introduction JPA

Les principaux avantages de JPA sont les suivants :

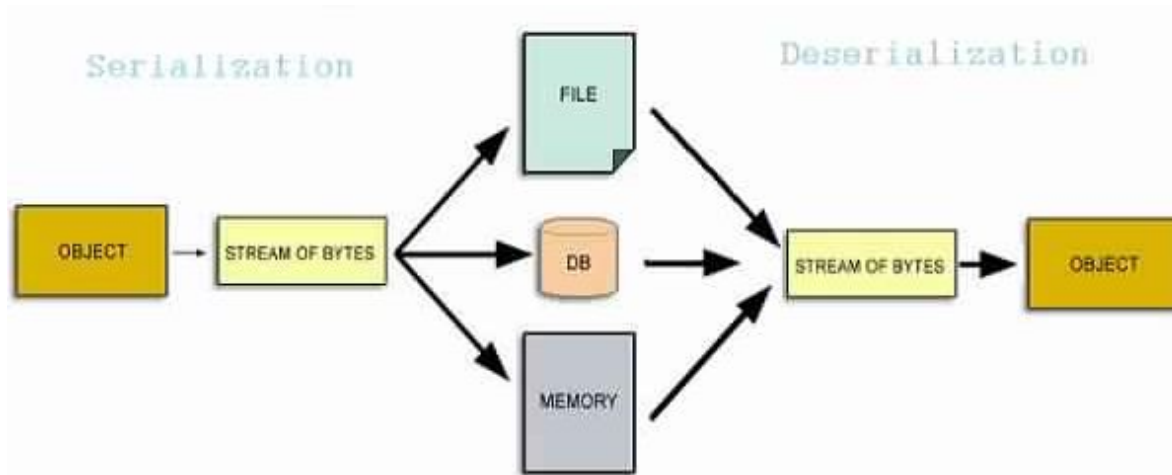
- JPA peut être utilisé par toutes les applications Java (Java SE ou Java EE).
- Mapping O/R (objet-relationnel) avec les tables de la BD, facilitée par les Annotations.
- Un langage de requête objet standard JPQL pour la récupération des objets.
- Possibilité de génération automatique du schéma de la base de données à partir des entités.

# La sérialisation

Le mécanisme de sérialisation consiste à convertir un objet en une suite de bits.

Nous pouvons ainsi sérialiser les objets sur disque, sur une connexion réseau (notamment Internet), sous un format indépendant des systèmes d'exploitation.

Java fournit un mécanisme simple, transparent et standard de sérialisation des objets via **l'implémentation** de l'interface `java.io.Serializable`.





# Les entités JPA

Une entité JPA est une classe JAVA qui doit satisfaire les conditions suivantes :

- Être déclaré avec l'annotation **@Entity**.
- Posséder au moins une propriété déclarée comme identité de l'entité avec l'annotation **@Id**.
- Implémenter l'interface `java.io.Serializable`.
- Tous les attributs doivent être « private » ou « protected » et avoir obligatoirement des accesseurs et des mutateurs (les méthodes **getters et setters**).

# Les entités JPA

Le fournisseur de persistance accédera à la valeur d'une variable d'instance:

- Soit en accédant directement à la variable d'instance
- Soit en passant par ses accesseurs (getter ou setter)

Le constructeur vide existe par défaut si aucun constructeur n'existe dans la classe.

```
@Entity
public class Employee implements Serializable {
    private Integer id;
    private String name;

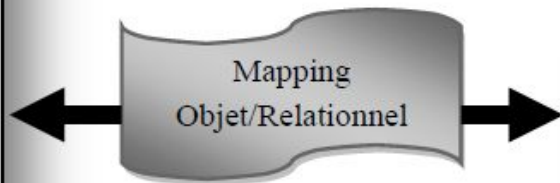
    public Employee() {
        super();
    }

    @Id
    public Integer getId() {
        return this.id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

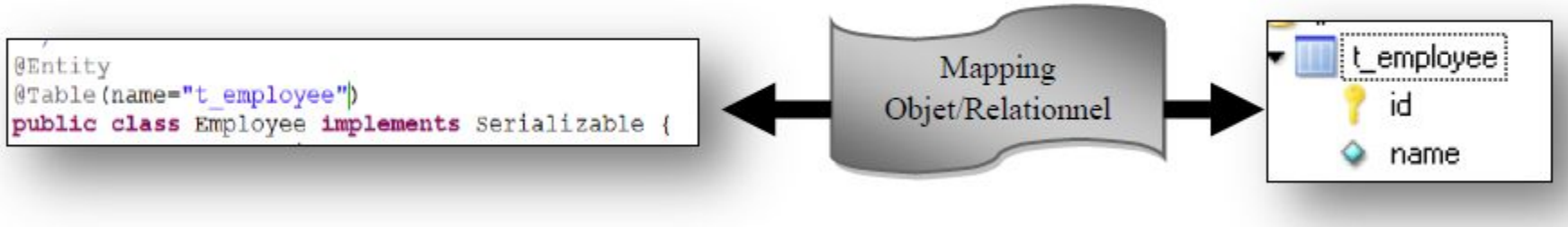
    public String getName() {
        return this.name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

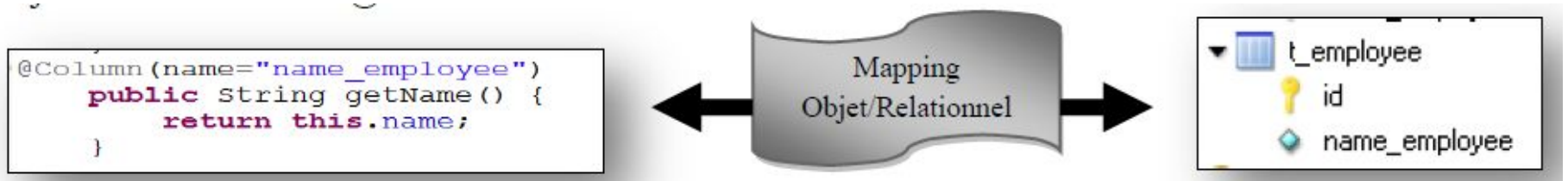


# Les entités JPA

Pour donner à la table un autre nom que le nom de la classe, il faut ajouter une annotation **@Table**



Pour donner à une colonne de la table un autre nom que le nom de l'attribut correspondant, il faut ajouter une annotation **@Column**



# Basic Types

- Java primitive types (boolean, int, etc)
- wrappers for the primitive types (java.lang.Boolean, java.lang.Integer, etc)
- java.lang.String
- java.math.BigInteger
- java.math.BigDecimal
- java.util.Date
- java.util.Calendar
- java.sql.Date
- java.sql.Time
- java.sql.Timestamp
- byte[] or Byte[]
- char[] or Character[]
- enums

[https://docs.jboss.org/hibernate/orm/5.2/userguide/html\\_single/Hibernate\\_User\\_Guide.html#basic](https://docs.jboss.org/hibernate/orm/5.2/userguide/html_single/Hibernate_User_Guide.html#basic)

# Les dates

Lorsqu'une entité jpa a un attribut de type temporel (Calendar ou Date de java.util), il est obligatoire d'indiquer de quel type temporel est cet attribut par une annotation `@Temporal`.

- **`TemporalType.DATE`** : la date va être sauvegardée dans la base de données sous forme d'une date (jour, mois, année)
- **`TemporalType.TIMESTAMP`** : la date va être sauvegardée dans la base de données sous forme d'une date (jour, mois, année) + heure à la microseconde.
- **`TemporalType.TIME`** : Un temps sur 24 heures (heures, minutes, secondes à la milliseconde près).

```
@Temporal(TemporalType.TIMESTAMP)
public Date getDateNaissance() {
    return dateNaissance;
}
```

Column Name	Datatype
DTYPE	VARCHAR(31)
id	INTEGER
dateNaissance	DATETIME

```
@Temporal(TemporalType.TIME)
public Date getDateNaissance() {
    return dateNaissance;
}
```

Column Name	Datatype
DTYPE	VARCHAR(31)
id	INTEGER
dateNaissance	TIME

```
@Temporal(TemporalType.DATE)
public Date getDateNaissance() {
    return dateNaissance;
}
```

Column Name	Datatype
DTYPE	VARCHAR(31)
id	INTEGER
dateNaissance	DATE

# Stratégies d'auto génération des IDs

Si l'ID est de type String, int, byte, char, double, float, long ou short, l'insertion de l'id peut se faire automatiquement par l'une des stratégies suivantes :

**IDENTITY** : La génération de la clé primaire se fait à partir d'une Identité **propre au SGBD**. Il utilise un type de colonne spéciale à la base de données.

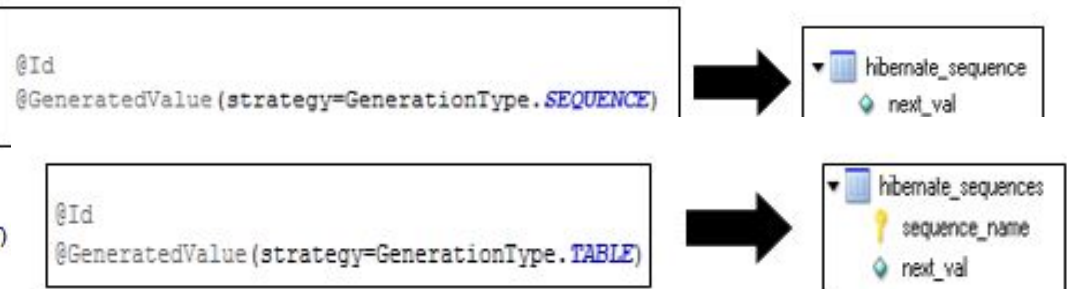
Exemple pour MySQL, il s'agit d'un **AUTO\_INCREMENT**.

**SEQUENCE** : la génération de la clé primaire est garantie par le fournisseur de persistance (hibernate), une séquence de clé primaire pour un schéma de base de données défini dans une table hibernate\_sequence.

**TABLE** : la génération des clés primaires est garanti par le fournisseur de persistance (hibernate), une séquence de clés primaires par table définie dans une table **hibernate\_sequences**, cette table contient deux colonnes : une pour le nom de la séquence et l'autre pour la prochaine valeur.

```
@Id
@GeneratedValue
private int id;
```

```
@Id
@GeneratedValue (strategy=GenerationType.IDENTITY)
```



# Les énumérations

**EnumType.String** : Si la valeur de l'attribut va être stockés dans la base de données sous forme de String qui représente la valeur de l'énumération.

**EnumType.Ordinal**: Si la valeur de l'attribut va être stockés dans la base de données sous forme de Integer qui représente la valeur de l'énumération.

```
public enum Domaine {  
  
    mathematiques, informatique, biologie  
  
}
```

```
@Enumerated(EnumType.STRING)  
public Domaine getDomaine() {  
    return domaine;  
}
```

```
@Enumerated(EnumType.ORDINAL)  
public Domaine getDomaine() {  
    return domaine;  
}
```

Column Name	Datatype
◆ salaire	◆ FLOAT
◆ specialite	◆ VARCHAR(255)
◆ domaine	◆ VARCHAR(255)
◆ grade	◆ VARCHAR(255)

Column Name	Datatype	NOT NULL
◆ salaire	◆ FLOAT	✓
◆ specialite	◆ VARCHAR(255)	
◆ domaine	◆ INTEGER	
◆ grade	◆ VARCHAR(255)	

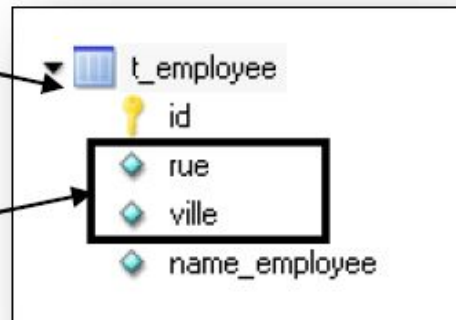
# Classe Embeddable

Il existe aussi des classes « insérées » ou « incorporées » (embedded) dont les données n'ont pas d'identité dans la BD mais sont insérées dans une des tables associées à une entité persistante

```
@Entity
@Table(name="t_employee")
public class Employee implements Serializable {
    private Integer id;
    private String name;
    private Adresse adresse;
    @Embedded
    public Adresse getAdresse() {
        return adresse;
    }
}
```

```
@Embeddable
public class Adresse implements java.io.Serializable {

    private String rue;
    private String ville;
}
```





# Clé composite

Pas recommandé, mais une clé primaire peut être composée de plusieurs colonnes.

- Peut arriver quand la BD existe déjà ou quand la classe correspond à une table d'association (association M:N avec information )

## 2 solutions :

-@IdClass

-@EmbeddedId et @Embeddable

Dans les 2 cas, la clé primaire doit être représentée par une classe Java dont les attributs correspondent aux composants de la clé primaire.

⇒ La classe doit être public, posséder un constructeur sans paramètre, être sérialisable et redéfinir **equals** et **hashCode**.

# Clé composite avec @EmbeddedId

@EmbeddedId correspond au cas où la classe entité comprend un seul attribut annoté @EmbeddedId

- La classe clé primaire est annoté par @Embeddable

```
@Entity
public class Employe {
    @EmbeddedId
    private EmployePK id;
    ...
}

@Embeddable
public class EmployePK {
    private String nom;
    private Date dateNaissance;
    ...
}
```

# Clé composite avec @IdClass

-@IdClass correspond au cas où la classe entité comprend plusieurs attributs annotés par @Id

La classe **entité** est annotée par **@IdClass** qui prend en paramètre le nom de la classe clé primaire

-La **classe clé primaire n'est pas annotée** (comme @Embeddable) ; ses attributs ont les mêmes noms et mêmes types que les attributs annotés @Id dans la classe entité

```
@Entity
@IdClass(EmployeePK.class)
public class Employe {
    @Id
    private String nom;
    @Id
    Date dateNaissance;
    ...
}

// Classe PK sans annotations
public class EmployeePK {
    private String nom;
    private Date dateNaissance;
    ...
}
```

# Héritage

Pour représenter un modèle hiérarchique dans un modèle relationnel, JPA propose alors trois stratégies possibles :

1. **Une seule table unique** pour l'ensemble de la hiérarchie des classes. L'ensemble des attributs de toute la hiérarchie des entités est mis à plat et regroupé dans une seule table (il s'agit d'ailleurs de la stratégie **par défaut**).

**@Inheritance(strategy=InheritanceType.SINGLE\_TABLE)**

2. **Une table pour chaque classe concrète**. Chaque entité concrète de la hiérarchie est associée à une table. **@Inheritance(strategy=InheritanceType.TABLE\_PER\_CLASS)**

3. **Jointure entre sous-classes**. Dans cette approche, chaque entité de la hiérarchie, concrète ou abstraite, est associée à sa propre table. Ainsi, nous obtenons dans ce cas une séparation des attributs spécifiques de la classe fille par rapport à ceux de la classe parente. Il existe alors, une table pour chaque classe fille, plus une table pour la classe parente.

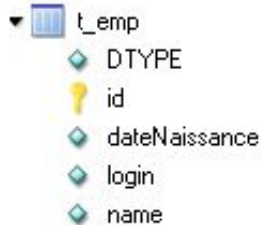
**@Inheritance(strategy=InheritanceType.JOINED)**

# Héritage single Table (par défaut)

La colonne Discriminator est gérée par JPA, certes une configuration personnalisée est permise pour le nom de la colonne discriminateur et la valeur de discrimination.

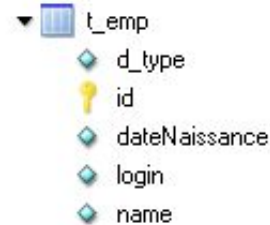
```
@Entity
@DiscriminatorValue(value="tech")
public class Technicien extends Employe implements Serializable {
```

```
@Entity
@Table(name = "t_emp")
public class Employe implements Serializable {
```



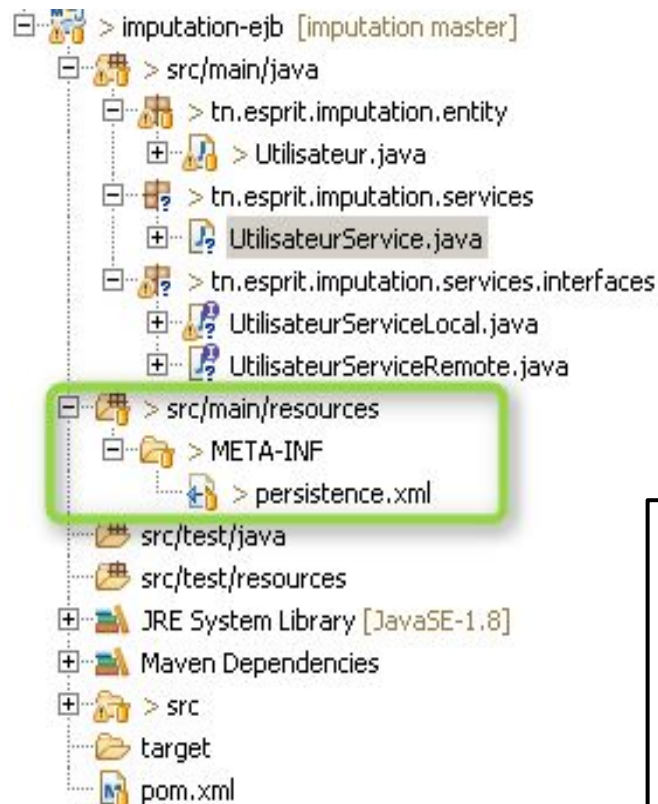
Configuration par défaut

```
@Entity
@Table(name = "t_emp")
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="d_type")
public class Employe implements Serializable {
```



Configuration personnalisée

# persistence.xml

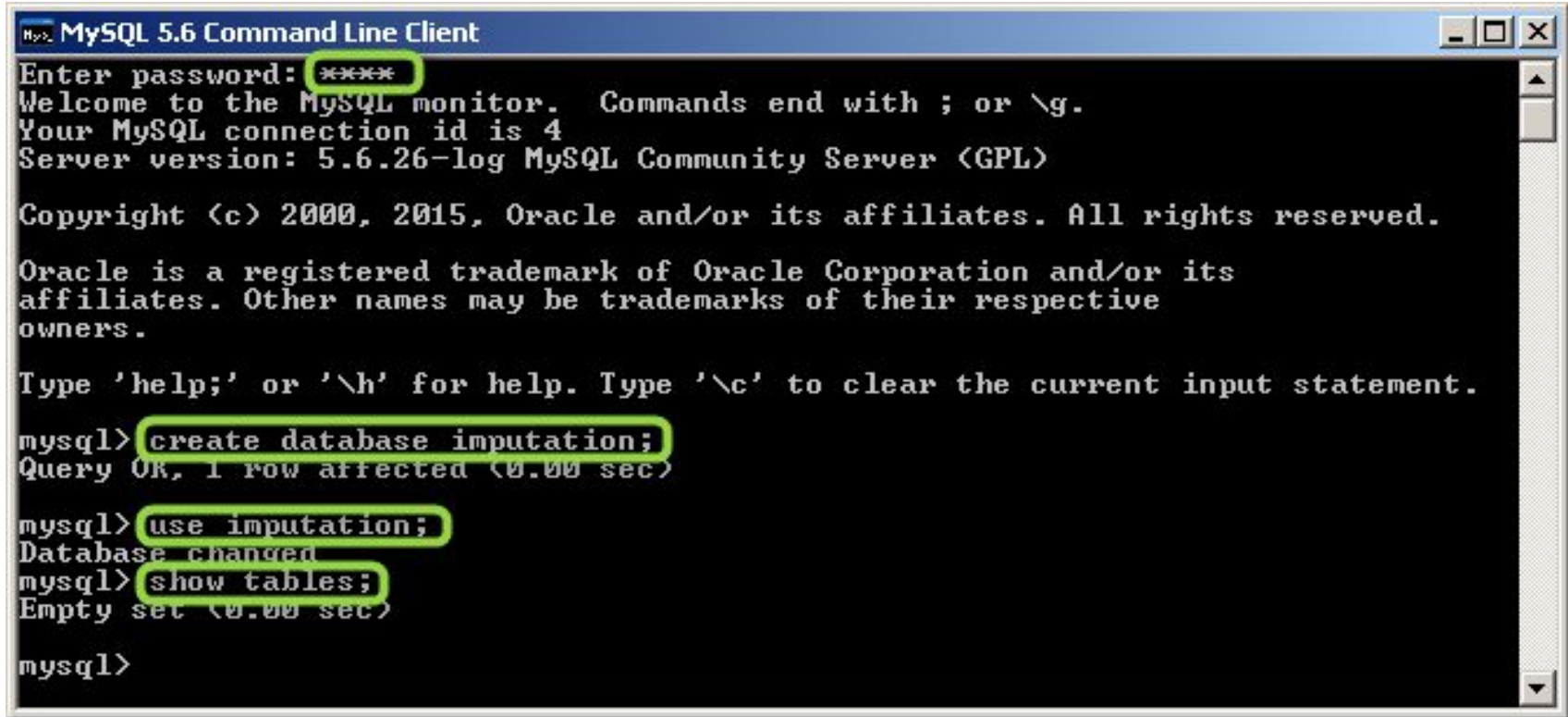


```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1"
  xmlns="http://xmlns.jcp.org/xml/ns/persistence" xmlns:xsi="http://
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence http://
  <persistence-unit name="imputation-ejb">
    <jta-data-source>java:/IMPUTATION_DS</jta-data-source>
    <properties>
      <property name="hibernate.hbm2ddl.auto" value="create" />
      <property name="hibernate.show_sql" value="true" />
      <property name="hibernate.format_sql" value="true" />
    </properties>
  </persistence-unit>
</persistence>
```

- Le fichier persistence.xml doit être stocké dans le répertoire META-INF
- Il contient un ou plusieurs tags <persistence-unit>
- Il contient les informations sur la connexion à la base de données à utiliser.

# Création de la première entité

- 1) Installer le serveur Mysql
- 2) Créer une base de données qui s'appelle "imputation"



```
MySQL 5.6 Command Line Client
Enter password: ****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4
Server version: 5.6.26-log MySQL Community Server (GPL)

Copyright (c) 2000, 2015, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

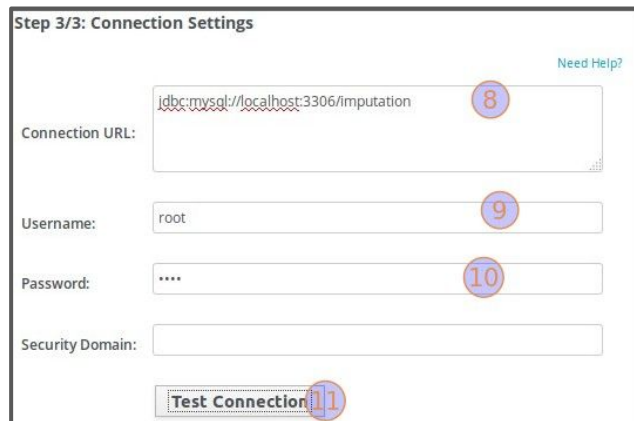
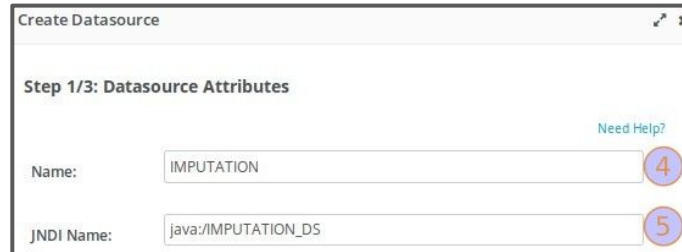
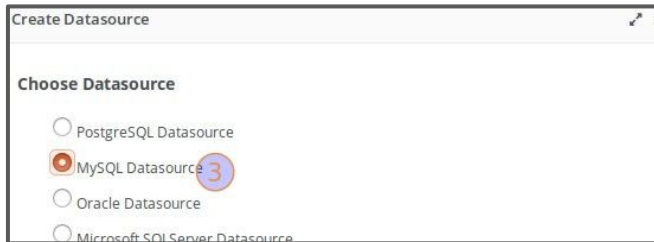
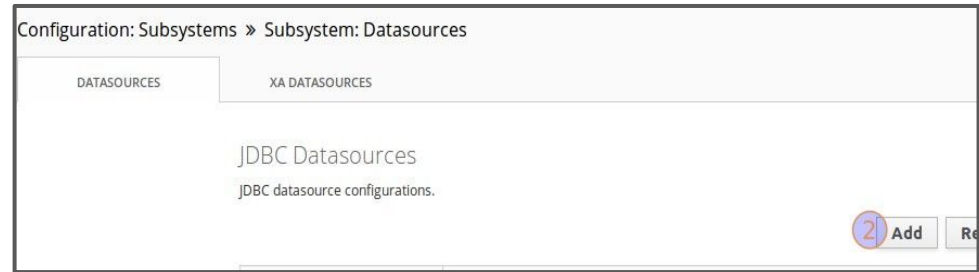
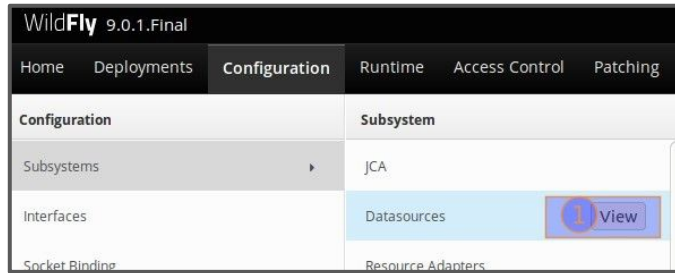
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> create database imputation;
Query OK, 1 row affected (0.00 sec)

mysql> use imputation;
Database changed
mysql> show tables;
Empty set (0.00 sec)

mysql>
```

# Création de la datasource <http://127.0.0.1:19990> (admin / wildflyadmin)





# Autre méthode pour créer la datasource

- 1) Aller au standalone.xml
- 2) Créer une nouvelle datasource "java:/IMPUTATION\_DS" qui soit un clone de "java:jboss/datasources/MySQLDS" (on change juste le jndi-name).
- 3) Mettre le bon user et mdp de votre serveur Mysql.
- 4) Mettre le nom de la base de données (base de donnée déjà crée "imputation")
- 5) Donner un nom au pool-name

```
<datasource jta="false" jndi-name="java:/IMPUTATION_DS" pool-name="IMPUTATION" er  
"false">  
  <connection-url>jdbc:mysql://localhost:3306/imputation</connection-url>  
  <driver-class>com.mysql.jdbc.Driver</driver-class>  
  <driver>mysql</driver>  
  <security>  
    <user-name>root</user-name>  
    <password>root</password>  
  </security>  
  <validation>  
    <validate-on-match>false</validate-on-match>  
    <background-validation>false</background-validation>  
  </validation>  
  <statement>  
    <share-prepared-statements>false</share-prepared-statements>  
  </statement>  
</datasource>
```

Nom de la base de  
données

# Création de la premiere entité

1) Importer l'archetype (en suivant la video)

Dans cette partie, on va travailler que sur le projet EJB :

2) Aller au projet EJB et ajouter le fichier persistence.xml sous <src/main/resources / META-INF> et changer le nom du datasource avec celle qu'on a créée "java:/IMPUTATION\_DS".

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1"
  xmlns="http://xmlns.jcp.org/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence http://xmlns.jcp.org/xml/ns/persistence/persistence.xml"
  <persistence-unit name="imputation-ejb">
    <jta-data-source>java:/IMPUTATION_DS</jta-data-source>
    <properties>
      <property name="hibernate.hbm2ddl.auto" value="update" />
      <property name="hibernate.show_sql" value="true" />
      <property name="hibernate.format_sql" value="true" />
    </properties>
  </persistence-unit>
</persistence>
```

3) Créer un package tn.esprit.imputation.entity sous <src/main/java>

4) Créer l'entité utilisateur avec les attributs nom, prenom et adresse email

5) déployer le projet sur Wildfly et voir les logs

# Création de la premiere entité

6) Dans les logs du serveur :

```
INFO [java.sql.DatabaseMetaData] (ServerService Thread Pool -- 58) HHH000262: Table not found: Utilisateur  
INFO [org.hibernate.tool.hbm2ddl.SchemaUpdate] (ServerService Thread Pool -- 58) HHH000232: Schema update complete
```

7) vérifier que la table Utilisateur a été bien créé dans la base de données "imputation" :

```
mysql> use imputation;  
Database changed  
mysql> show tables;  
+-----+  
| Tables_in_imputation |  
+-----+  
| utilisateur           |  
+-----+  
1 row in set (0.00 sec)  
  
mysql> describe utilisateur;  
+-----+-----+-----+-----+-----+-----+  
| Field | Type          | Null | Key | Default | Extra |  
+-----+-----+-----+-----+-----+-----+  
| id    | int(11)       | NO   | PRI | NULL    |       |  
| email | varchar(255)  | YES  |     | NULL    |       |  
| nom   | varchar(255)  | YES  |     | NULL    |       |  
| prenom | varchar(255)  | YES  |     | NULL    |       |  
+-----+-----+-----+-----+-----+-----+  
4 rows in set (0.00 sec)
```