



Optez pour une transformation digitale intelligente

L'INTELLIGENCE ARTIFICIELLE AU SERVICE DE VOTRE BUSINESS.



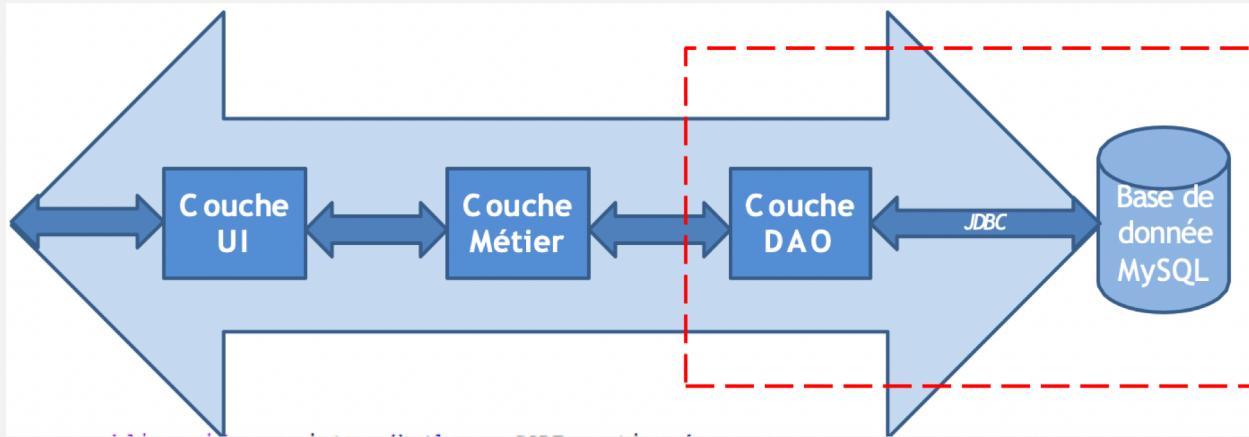
JPA



Programme

- Historique
- ORM : Object Relational Mapping
- JPA : JAVA Persistence API
- Entity: Entités
- Autres annotations
- Contexte de persistance
- Opérations prises en charge par le gestionnaire d'entités
- Cycle de vie d'une instance d'entité
- Obtention d'une fabrique EntityManagerFactory
- Création du gestionnaire d'entité EntityManager
- Exemple d'insertion d'un livre
- Relations entre entités

Historique : Accès directement à la base de donnée grâce à l'API standard JDBC de Java



```
public void enregistrer() throws SQLException {
    //ouverture de la connexion jdbc
    Class.forName("com.mysql.jdbc.driver");
    String url = "jdbc:mysql://localhost/Bdbanque";
    Connection con = DriverManager.getConnection(url, "login", "password");

    //preparation ou construction de la requête sql
    String insertStatement = "Insert into Client(Nom, Prenom,Nature) values (?, ?, ?)";
    PreparedStatement prepStm1 = con.prepareStatement(insertStatement);
    prepStm1.setString(1,txtNom.getText());
    prepStm1.setString(2,txtPrenom.getText());
    prepStm1.setString(3,txtAge.getText());
    //execution de la requête
    prepStm1.executeUpdate();
    prepStm1.close();

    //fermeture de la connexion jdbc
    con.close();
}
```

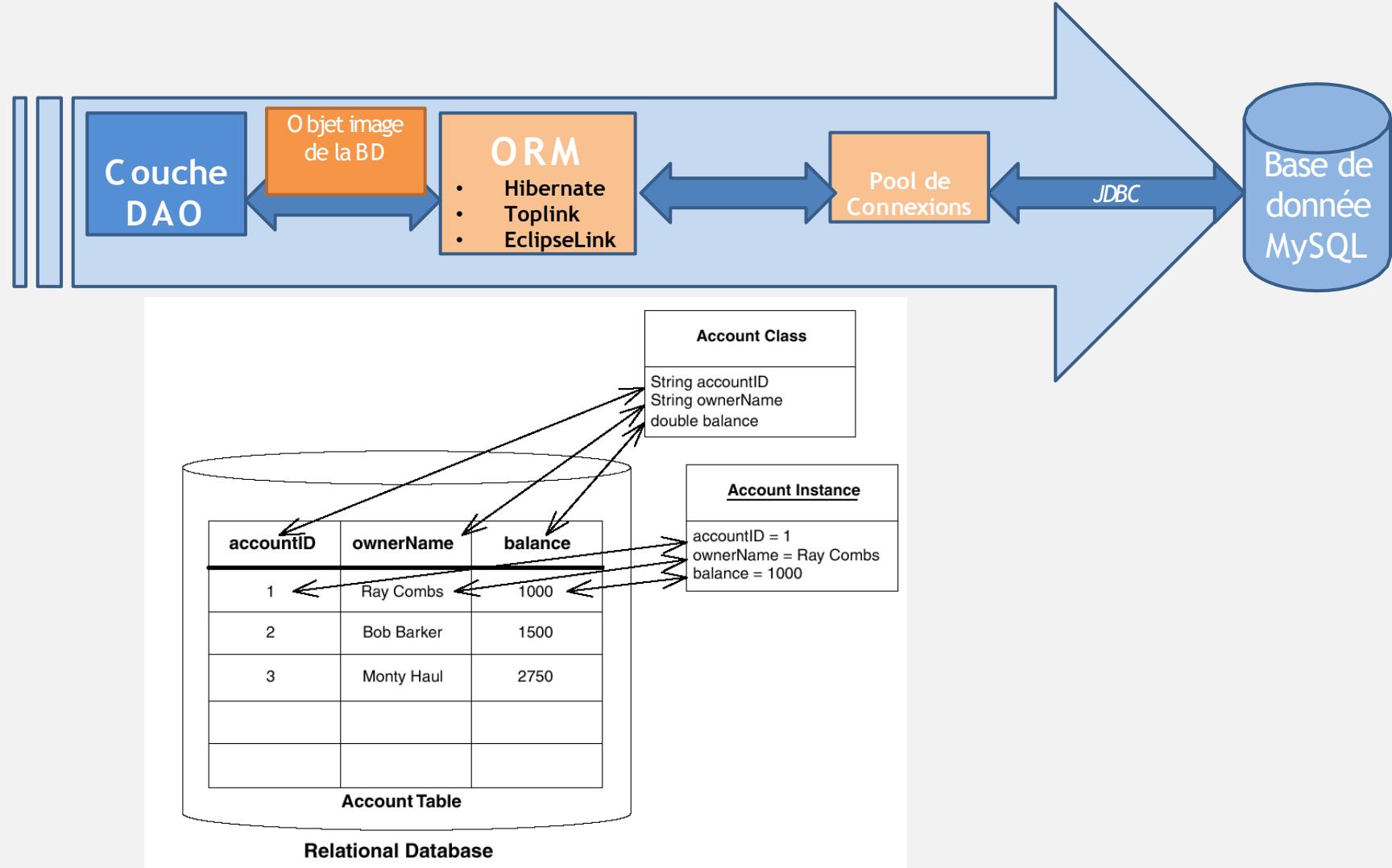
Problématiques de l'accès direct à la BD à l'aide de jdbc :

- Pour des raisons de performances, le coût d'ouverture / fermeture d'une connexion n'est pas négligeable,
 - `Connection con = DriverManager.getConnection(url, "login", "password");`
 - `//les ordres SQL`
 - `con.close();`
- L'utilisation du langage SQL rend la couche DAO difficilement maintenable,
 - `String insertStatement = "Insert into Client(Nom, Prenom,Nature) values (?,?,?)"`

Pour pallier à cette problématique et faciliter l'écriture de la couche DAO,

La communauté Java a donc fait naître des **Frameworks ORM** , tels que **Hibernate, Toplink, EclipseLink**

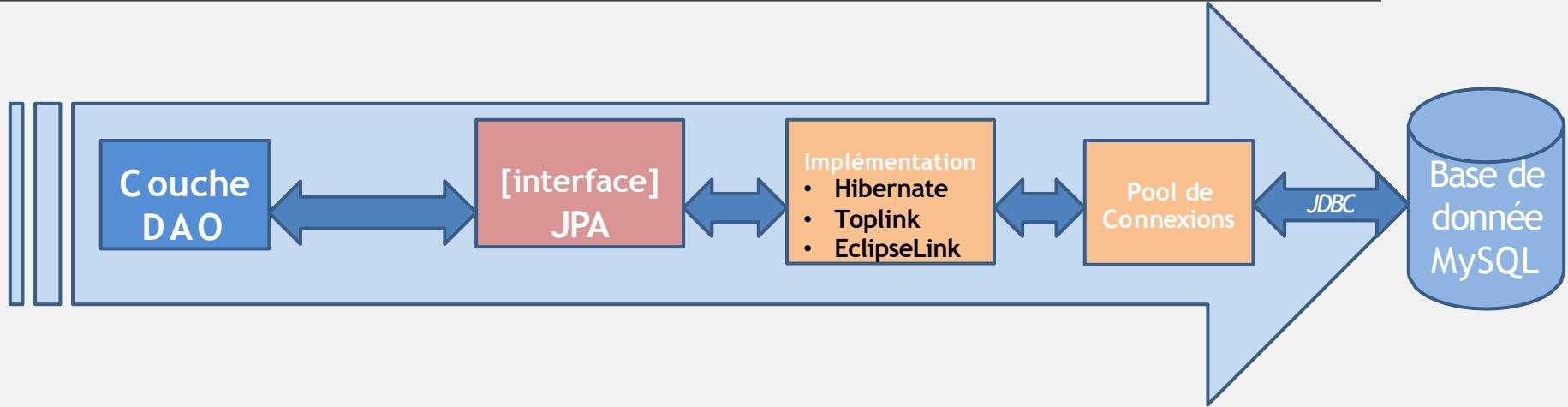
ORM : Object Relational Mapping



Devant le succès des Frameworks ORM,

Sun a décidé de standardiser une couche ORM via une spécification appelée JPA apparue en même temps que Java 5

JPA : Java Persistence Api



La spécification JPA est un ensemble d'**interface** du package [javax.persistence](#)

Exemple :

```
javax.persistence.Entity;  
javax.persistence.EntityManagerFactory;  
javax.persistence.EntityManager;  
javax.persistence.EntityTransaction;
```

[EntityManager]
void persist(Entity entité)

JPA : Java Persistence Api

Ses principaux avantages sont les suivants :

1. JPA peut être utilisé par toutes les applications **Java, Java SE ou Java EE.**
2. Mapping O /R (objet-relationnel) avec les tables de la BD, facilitée par les Annotations.
select p from Personne p order by p.nom asc
3. Un langage de requête objet standard **JPQL** pour la récupération des objets,

Entity: Entités

```
@Entity
@Table(name = "BOOK")
public class Book {
    @Id
    @GeneratedValue
    private Long id;
    @Column(name = "TITLE", nullable = false)
    private String title;
    private Float price;
    @Basic(fetch = FetchType.LAZY)
    @Column(length = 2000)
    private String description;
    private String isbn;
    private Integer nbOfPage;
    private Boolean illustrations
```

// Les Getters et les Setters

<<entity>>	
Book	
-id : Long	
-title : String	
-price : Float	
-description : String	
-isbn : String	
-nbOfPage : Integer	
-illustrations : Boolean	

← mapping →

BOOK		
+ID	bigint	Nullable = false
TITLE	varchar(255)	Nullable = false
PRICE	double	Nullable = true
DESCRIPTION	varchar(2000)	Nullable = true
ISBN	varchar(255)	Nullable = true
NBOFPAGE	integer	Nullable = true
ILLUSTRATIONS	smallint	Nullable = true

Contexte de persistance

Ensemble des instances d'entités **gérées** à un instant donné,

- **gérées par qui ?**
- Le gestionnaire d'entités :**EntityManager**
- Ce contexte peut donc être considéré comme **un cache de premier niveau** : c'est un espace réduit où le gestionnaire stocke les entités avant d'écrire son contenu dans la base de données,

Les Entités

- Les classes dont les instances peuvent être persistantes sont appelées des entités dans la spécification de JPA
 - Le développeur indique qu'une classe est une entité en lui associant l'annotation `@Entity`
 - Il faut importer `javax.persistence.Entity` dans les classes entités.
- Les entités dans les spécifications de l'API Java Persistence permettent d'encapsuler les données d'une occurrence d'une ou plusieurs tables.
- Ce sont de simples POJO
 - Un POJO est une classe Java qui n'implémente aucune interface particulière ni n'hérite d'aucune classe mère spécifique.

Classes entités : Entity Class

- **Une classe entité doit posséder un attribut qui représente la clé primaire dans la BD (@Id)**
 - Elle doit avoir un constructeur sans paramètre **protected** ou **public** sans argument et la classe du bean doit obligatoirement être marquée avec l'annotation **@javax.persistence.Entity**. Cette annotation possède un attribut optionnel nommé **name** qui permet de préciser le nom de l'entité dans les requêtes. Par défaut, ce nom est celui de la classe de l'entité.
 - Elle ne doit pas être final
 - Aucune méthode ou champ persistant ne doit être final
 - Une entité peut être une classe abstraite mais elle ne peut pas être une interface

Les annotations

Annotation	Rôle
@javax.persistence.Table	Préciser le nom de la table concernée par le mapping
@javax.persistence.Column	Associé à un getter, il permet d'associer un champ de la table à la propriété
@javax.persistence.Id	Associé à un getter, il permet d'associer un champ de la table à la propriété en tant que clé primaire
@javax.persistence.GeneratedValue	Demander la génération automatique de la clé primaire au besoin
@javax.persistence.Basic	Représenter la forme de mapping la plus simple. Cette annotation est utilisée par défaut
@javax.persistence.Transient	Demander de ne pas tenir compte du champ lors du mapping

L'annotation `@javax.persistence.Table`

- Permet de lier l'entité à une table de la base de données. Par défaut, l'entité est liée à la table de la base de données correspondant au nom de la classe de l'entité. Si ce nom est différent alors l'utilisation de l'annotation `@Table` est obligatoire.

Attributs	Rôle
<code>name</code>	Nom de la table
<code>catalog</code>	Catalogue de la table
...	...

L'annotation `@javax.persistence.Column`

- Permet d'associer un membre de l'entité à une colonne de la table. Par défaut, les champs de l'entité sont liés aux champs de la table dont les noms correspondent. Si ces noms sont différents alors l'utilisation de l'annotation **@Column** est obligatoire.

Attributs <code>name</code>	Rôle Nom de la colonne
<code>table</code>	Nom de la table dans le cas d'un mapping multi-table
<code>unique</code>	Indique si la colonne est unique
<code>Nullable</code>	Indique si la colonne est nullable
<code>insertable</code>	Indique si la colonne doit être prise en compte dans les requêtes de type insert
<code>updatable</code>	Indique si la colonne doit être prise en compte dans les requêtes de type update

L'annotation @Id

- **Il faut obligatoirement** définir une des propriétés de la classe avec l'annotation **@Id** pour la déclarer comme étant la clé primaire de la table.
- Cette annotation peut marquer soit **le champ** de la classe concernée soit **le getter** de la propriété.
- L'utilisation de l'un ou l'autre précise au gestionnaire s'il doit se baser sur les champs ou les getter pour déterminer les associations entre l'entité et les champs de la table. La clé primaire peut être constituée d'une seule propriété ou composées de plusieurs propriétés qui peuvent être de type primitif ou chaîne de caractères

EXEMPLE



A screenshot of a Java code editor displaying a class named `Personne`. The code uses annotations from the `javax.persistence` package to define an entity with an auto-generated ID and properties for prenom and nom.

```
import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

@Entity
public class Personne implements Serializable {

    @Id @GeneratedValue
    private int id;

    private String prenom;
    private String nom;
    private static final long serialVersionUID = 1L;

    public Personne() {
        super();
    }
    public int getId() {
        return this.id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getPrenom() {
        return this.prenom;
    }
    public void setPrenom(String prenom) {
        this.prenom = prenom;
    }
    public String getNom() {
        return this.nom;
    }
    public void setNom(String nom) {
        this.nom = nom;
    }
}
```

Cas d'une Clé primaire composée

- Le modèle de base de données relationnelle permet la définition d'une clé primaire composée de plusieurs colonnes.
- L'API Java Persistence propose deux façons de gérer ce cas de figure :
 - L'annotation `@javax.persistence.IdClass`
 - L'annotation `@javax.persistence.EmbeddedId`

L'annotation @javax.persistence.IdClass

- L'annotation @IdClass s'utilise avec une classe qui va encapsuler les propriétés qui composent la clé primaire. Cette classe doit obligatoirement :
 - Être sérialisable
 - Posséder un constructeur sans arguments
 - Fournir une implémentation dédiée des méthodes **equals()** et **hashCode()**

Exemple: PersonnePK (1/3)

```
public class PersonnePK implements java.io.Serializable {  
    private static final long serialVersionUID = 1L;  
    private String nom; private String prenom;  
  
    public PersonnePK() {}  
  
    public PersonnePK(String nom, String prenom) {  
        this.nom = nom; this.prenom = prenom;  
    }  
  
    public String getNom() {  
        return this.nom;  
    }  
    public void setNom(String nom) {  
        this.nom = nom;  
    }  
    public String getPrenom() {  
        return prenom;  
    }  
    public void setPrenom(String prenom) {  
        this.prenom = prenom;  
    }  
    public boolean equals(Object obj) {  
  
        boolean resultat = false;  
        if (obj == this) {  
            resultat = true;  
        }else {  
            if (!(obj instanceof PersonnePK)) {  
                resultat = false;  
            }else {  
                PersonnePK autre = (PersonnePK) obj;  
  
                if (!nom.equals(autre.nom)) {  
                    resultat = false;  
                }else {  
                    if (prenom != autre.prenom) {  
                        resultat = false;  
                    }else {  
                        resultat = true;  
                    }  
                }  
            }  
        }  
        return resultat;  
    }  
  
    public int hashCode() {  
        return (nom + prenom).hashCode()  
    }  
}
```

Exemple: (2/3)

Il est nécessaire de définir la classe de la clé primaire dans le fichier de configuration persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  version="1.0" xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence\_1\_0.xsd">
  <persistence-unit name="MaBaseDeTestPU">
    <provider>oracle.toplink.essentials.PersistenceProvider</provider>
    <class>com.jmd.test.jpa.Personne</class>
    <class>com.jmd.test.jpa.PersonnePK</class>
  </persistence-unit>
</persistence>
```

Exemple (3/3)

- Il faut utiliser l'annotation `@IdClass` sur la classe de l'entité.
- Il est nécessaire de marquer chacune des propriétés de l'entité qui compose la clé primaire avec l'annotation `@Id`.
- Ces propriétés doivent avoir le même nom dans l'entité et dans la classe qui encapsule la clé primaire.

```
import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.IdClass;

@Entity
@IdClass(PersonnePK.class)
public class Personne implements Serializable {

    private String prenom;
    private String nom;
    private int taille;
    private static final long serialVersionUID = 1L;

    public Personne() {
        super();
    }

    @Id
    public String getPrenom() {
        return this.prenom;
    }
    public void setPrenom(String prenom) {
        this.prenom = prenom;
    }
    @Id
    public String getNom() {
        return this.nom;
    }
    public void setNom(String nom) {
        this.nom = nom;
    }
    public int getTaille() {
        return this.taille;
    }
    public void setTaille(int taille) {
        this.taille = taille;
    }
}
```

Classes entités : Embedded Class

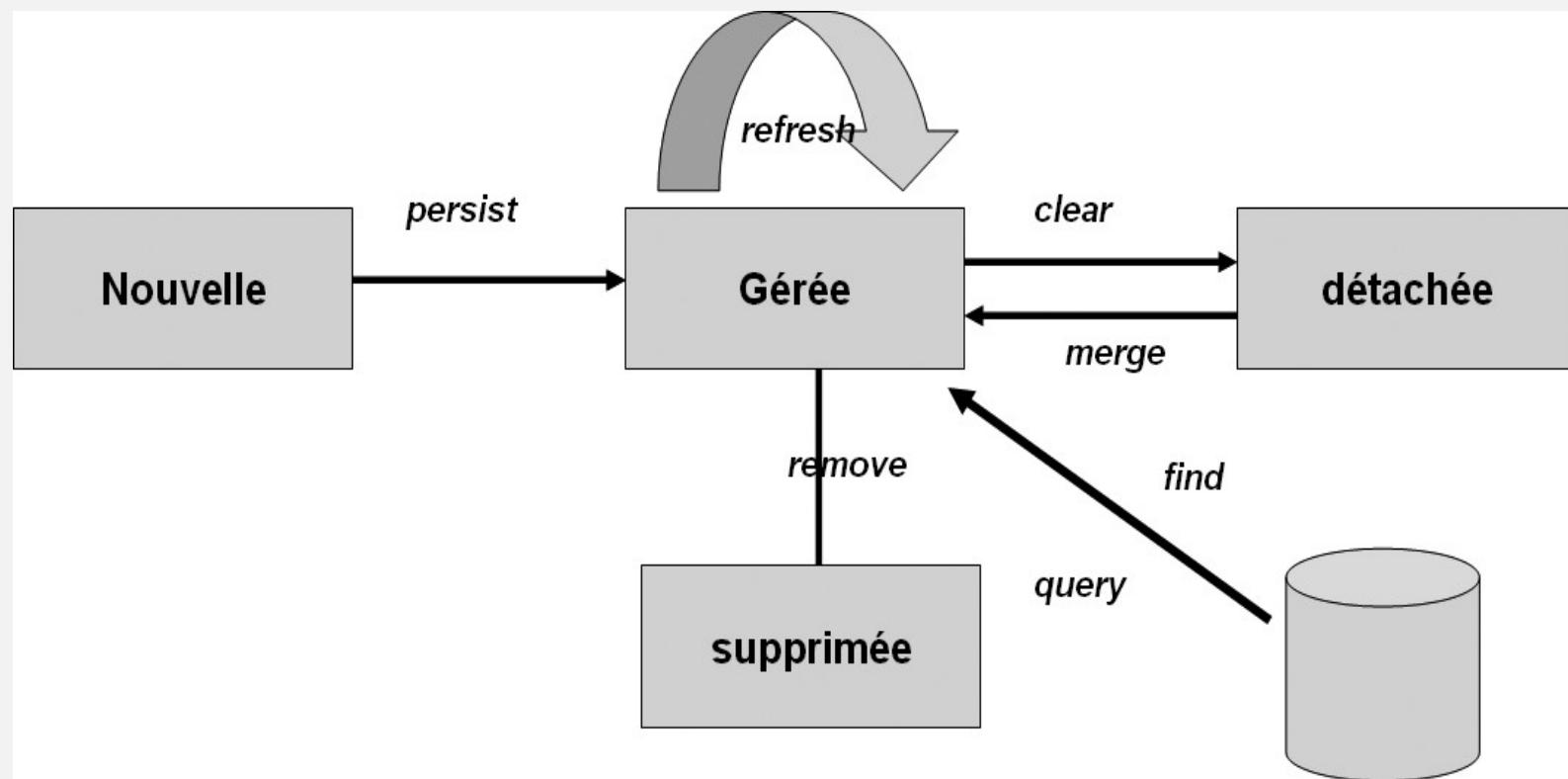
- Il existe des classes incorporées (*embedded*) dont les données n'ont pas d'identité dans la BD mais sont insérées dans une des tables associées à une entité persistante
 - – Par exemple, une classe Adresse dont les valeurs sont insérées dans la table Employe

Opérations prises en charge par le gestionnaire d'entité

Opération	Description
persist()	Insère l'état d'une entité dans la base de données. Cette nouvelle entité devient alors une entité gérée.
remove()	Supprime l'état de l'entité gérée et ses données correspondantes de la base.
refresh()	Synchronise l'état de l'entité à partir de la base, les données de la BD étant copiées dans l'entité.
merge()	Synchronise les états des entités « détachées » avec le PC. La méthode retourne une entité gérée qui a la même identité dans la base que l'entité passée en paramètre, bien que ce ne soit pas le même objet.
find()	Exécute une requête simple de recherche de clé.
CreateQuery()	Crée une instance de requête en utilisant le langage JPQL.
createNamedQuery()	Crée une instance de requête spécifique.
createNativeQuery()	Crée une instance de requête SQL.
contains()	Spécifie si l'entité est managée par le PC.
flush()	Toutes les modifications effectuées sur les entités du contexte de persistance gérées par le gestionnaire d'entités sont enregistrées dans la BD lors d'un flush du gestionnaire.

N B : Un flush() est automatiquement effectué au moins à chaque **commit de la transaction** en cours,

Cycle de vie d'une instance d'entité



Obtention d'une fabrique EntityManagerFactory

L' interface **EntityManagerFactory** permet d'obtenir une instance de l'objet **EntityManager**,

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("jpa");
```

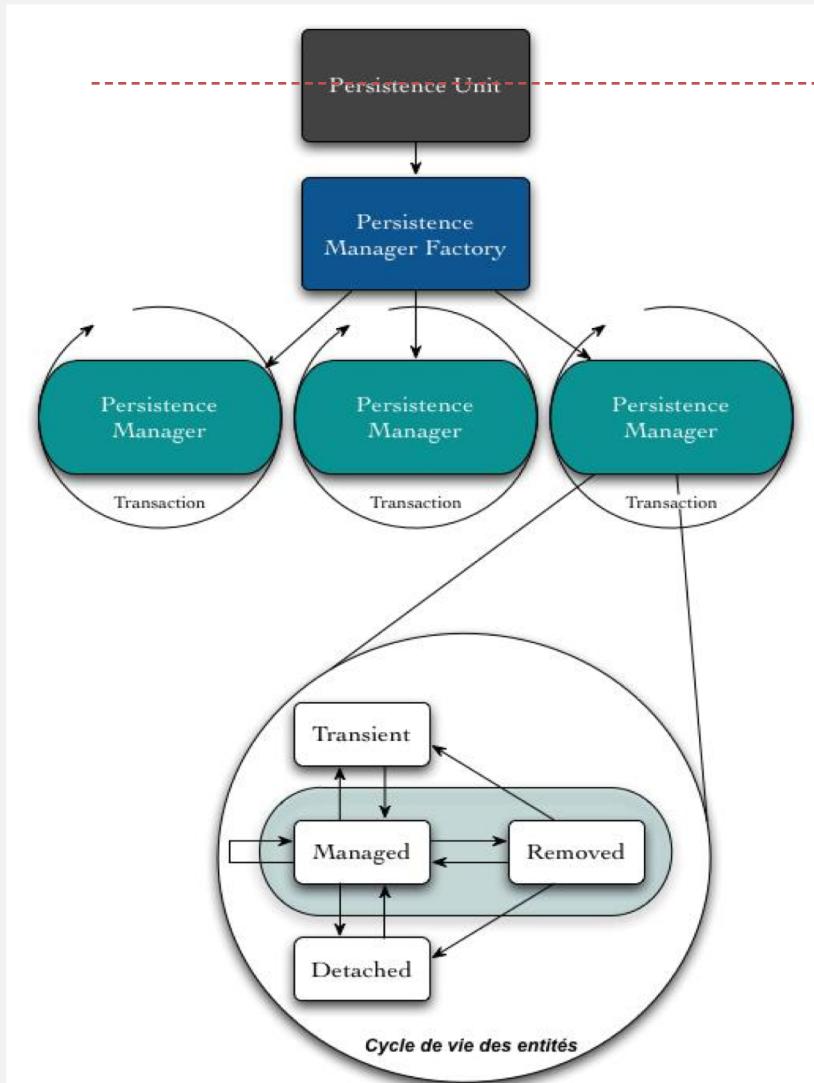
« **jpa** » est le nom de l'unité de persistance, définie dans le fichier de configuration de la couche JPA **META-INF/persistence.xml**.

Création du gestionnaire d'entité EntityManager

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("jpa");
```

```
EntityManager em = emf.createEntityManager();
```

Fonctionnement de JPA



- La **Persistence Unit** : organise les meta données qui définissent le mapping entre les entités et la base de donnée dans le fichier de configuration **META-INF/persistence.xml**
- La **Persistence Manager Factory** récupère ces metas données de la Persistence Unit et les interprètent pour créer des Persistence Manager (EntityManager)
- Le **Persistence Manager** (EntiyManager) gère les échanges entre le code et la base de donnée, c'est à dire le cycle de vie des **entités**
- Enfin, les opérations du EntityManager est englobé dans une **Transaction**.