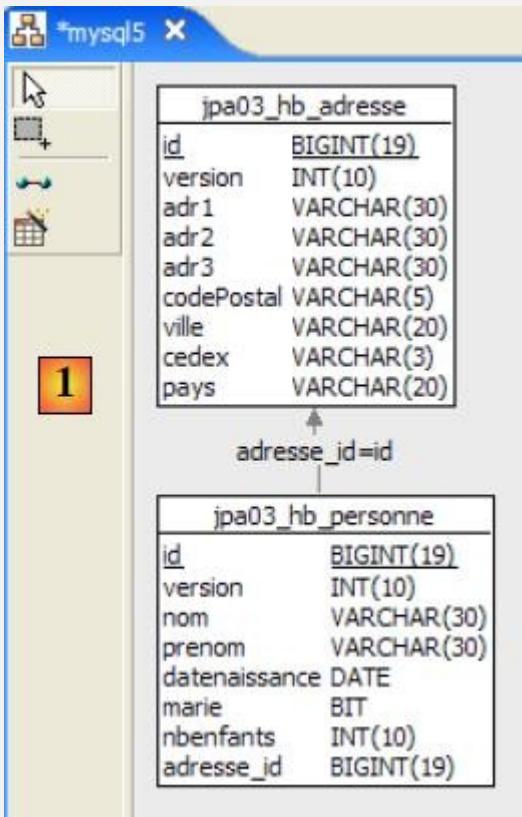


Relations

- Dans le modèle des bases de données relationnelles, les tables peuvent être liées entre elles grâce à des relations.
- Ces relations sont transposées dans les liaisons que peuvent les différentes entités correspondantes.
- **4 types de relations à définir entre les entités de la JPA:**
 - 1-1 (One to One)
 - 1-n (Many to One)
 - 1-n (One to Many)
 - n-n (Many to Many)

Relations entre entités

Relation un-à-un



```
@Entity
@Table(name = "jpa03_hb_personne")
public class Personne {
    ...
    @OneToOne(cascade = CascadeType.ALL, fetch=FetchType.LAZY)
    /*@OneToOne(cascade = {CascadeType.MERGE, CascadeType.PERSIST,
    CascadeType.REFRESH, CascadeType.REMOVE}, fetch=FetchType.LAZY) */
    @JoinColumn(name = "adresse_id", unique = true, nullable = false)
    private Adresse adresse;
    ...
}
```

```
@Entity
@Table(name = "jpa03_hb_adresse")
public class Adresse{
    ...
    /*n'est pas obligatoire*/
    @OneToOne(mappedBy = "adresse", fetch=FetchType.EAGER)
    private Personne personne;
    ...
}
```

Relationships: One to One

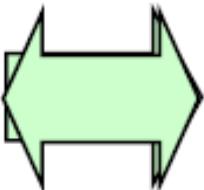
```
@Entity  
@Table(name="EMP")  
public class Employee {  
    @Id  
    private int id;
```

```
@OneToOne(name="P_SPACE")  
  
private ParkingSpace space;
```

```
// getters & setters
```

EMP ...

ID	P_SPACE		
PK	FK		



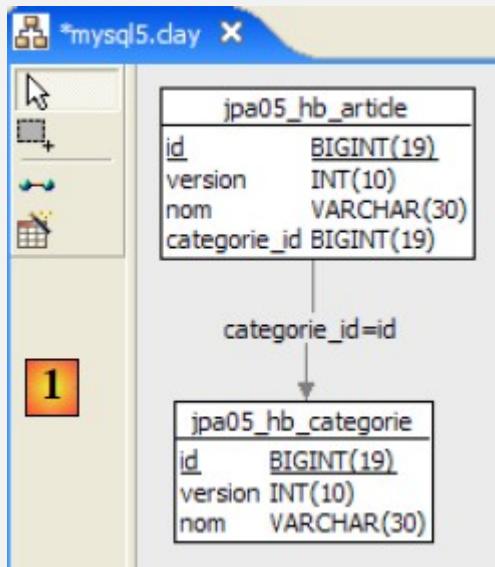
```
@Entity  
public class ParkingSpace {  
  
    @Id  
    private int id;  
  
    private int lot;  
  
    private String location;  
  
    @OneToOne(mappedBy="space")  
    private Employee emp;  
  
    // getters & setters  
    ...  
}
```

PARKINGSPACE

ID	LOT	LOCATION	
PK			

Relations entre entités

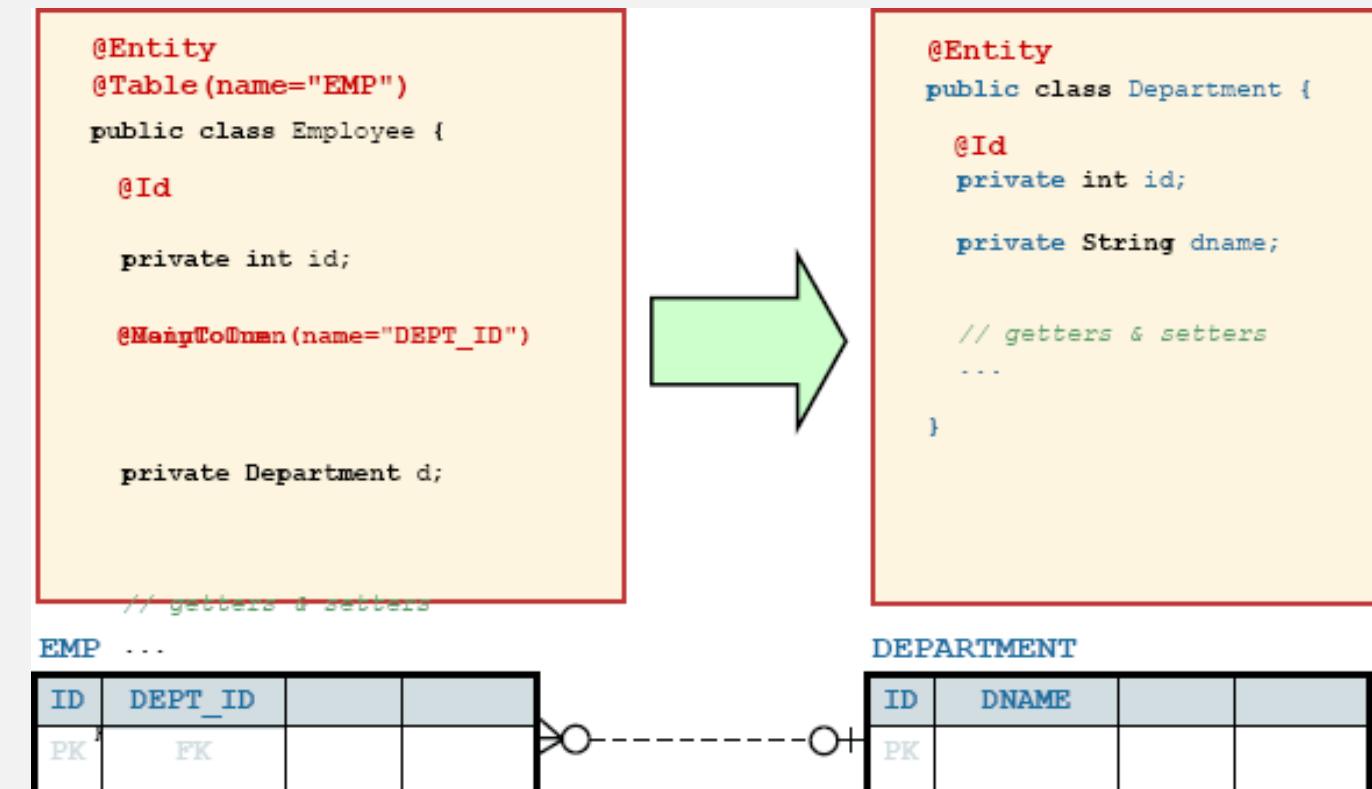
Relation un-à-plusieurs et plusieurs-à-un



```
@Entity  
@Table(name="jpa05_hb_article")  
public class Article{  
...  
// relation principale Article (many) -> Category (one) implémentée par une clé  
// étrangère (categorie_id) dans Article  
// 1 Article a nécessairement 1 Categorie (nullable=false)  
  
@ManyToOne(fetch=FetchType.LAZY)  
@JoinColumn(name = "categorie_id", nullable = false)  
private Categorie categorie;  
...  
}
```

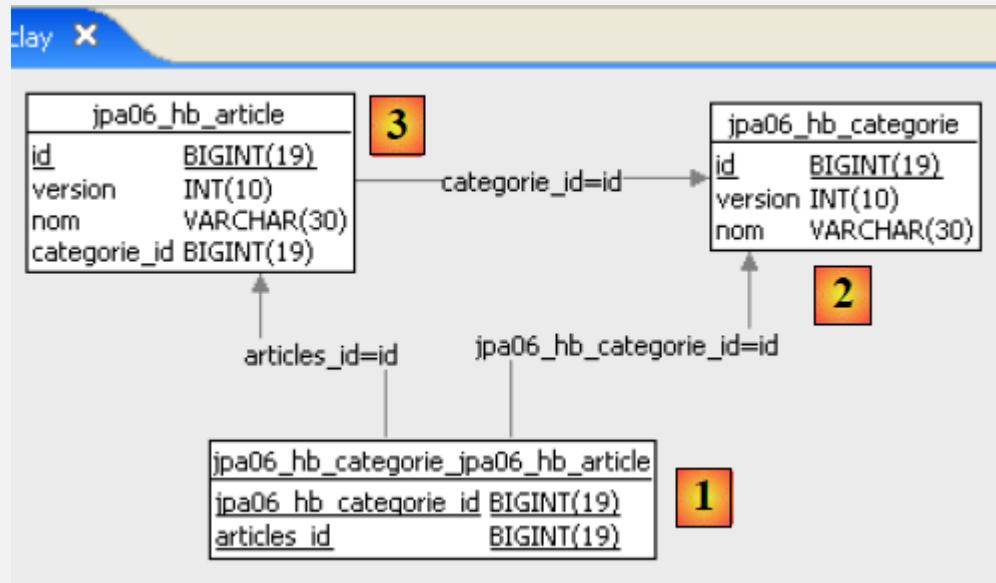
```
@Entity  
@Table(name="jpa05_hb_categorie")  
public class Categorie{  
...  
// relation inverse Categorie (one) -> Article (many) de la relation Article (many)-> Categorie  
(one)  
@OneToMany(mappedBy = "categorie", cascade = { CascadeType.ALL })  
private Set<Article> articles = new HashSet<Article>();  
...  
}
```

Relationship: Many to One



Relations entre entités

Relation un-à-plusieurs et plusieurs-à-un

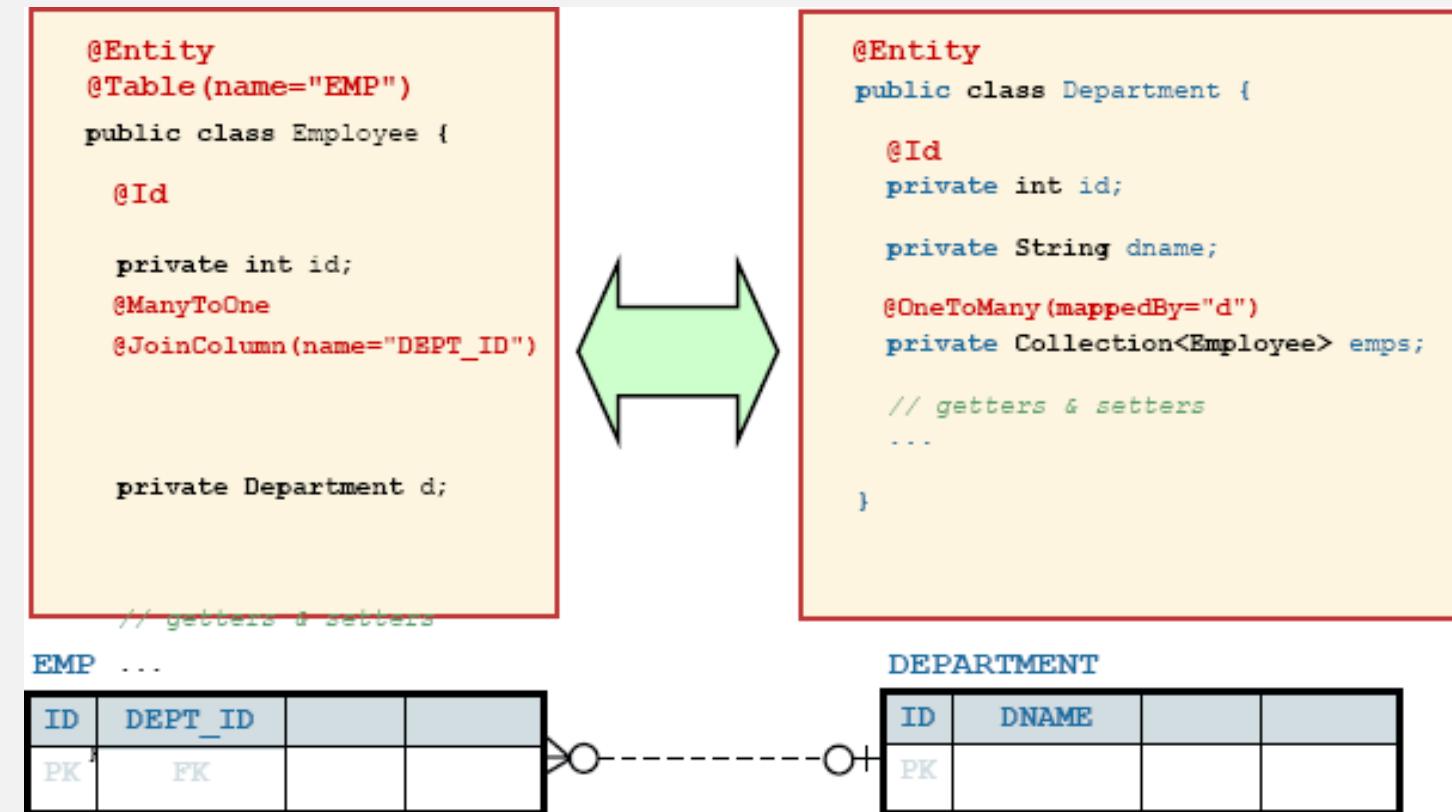


```
play X

@Entity @Table(name="jpa06_hb_categorie")
public class Categorie{
    ...
    // relation OneToMany non inverse (absence de mappedBy) Categorie (one) -> Article (many)
    // implémentée par une table de jointure Categorie_Article pour qu'à partir d'une catégorie
    // on puisse atteindre plusieurs articles

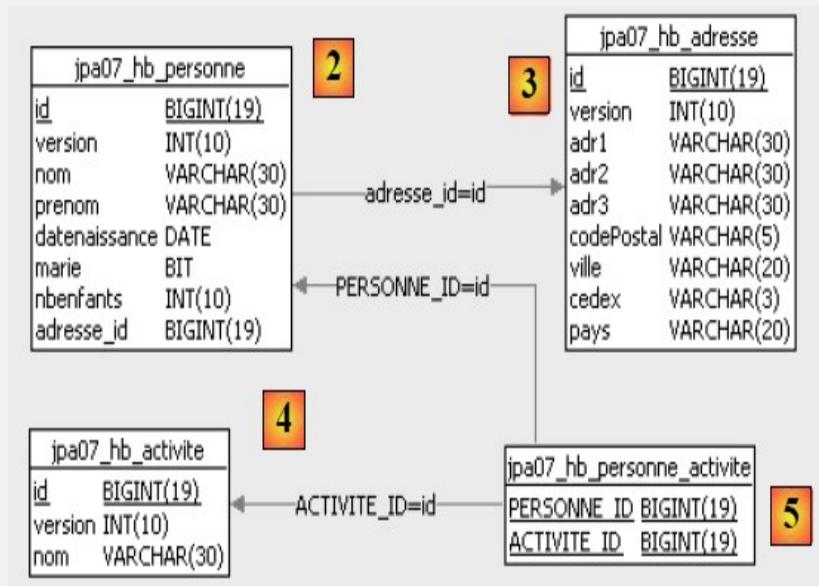
    @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY)
    private Set<Article> articles = new HashSet<Article>();
    ...
}
```

Relationship: One to Many



Relations entre entités

Relation plusieurs-à-plusieurs



```
@Entity
@Table(name = "jpa07_hb_personne")
public class Personne {
    ...
    // relation Personne (many) -> Activite (many) via une table de jointure personne_activite

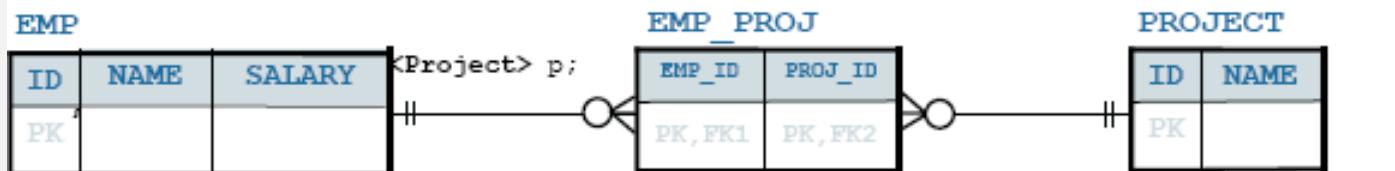
    @ManyToMany(cascade={CascadeType.PERSIST})
    @JoinTable(name="jpa07_hb_personne_activite", joinColumns = @JoinColumn(name = "PERSONNE_ID"),
    inverseJoinColumns = @JoinColumn(name = "ACTIVITE_ID"))
    private Set<Activite> activites = new HashSet<Activite>();
    ...
}
```

```
@Entity
@Table(name = "jpa07_hb_activite")
public class Activite {
    ...
    // relation inverse Activite -> Personne @ManyToMany(mappedBy = "activites")
    private Set<Personne> personnes = new HashSet<Personne>();
    ...
}
```

Relationships: Many to Many

```
@Entity  
@Table(name="EMP")  
public class Employee {  
  
    @Id  
  
    private int id;  
  
    @JoinTable(name="EMP_PROJ",  
              joinColumns=  
                  @JoinColumn(name="EMP_ID"),  
              inverseJoinColumns=  
                  @ManyToManyColumn(name="PROJ_ID"))
```

```
@Entity  
public class Project {  
  
    @Id  
    private int id;  
  
    private String name;  
  
    @ManyToMany(mappedBy="p")  
    private Collection<Employee> e;  
  
    // getters & setters  
    ...  
}
```



EntityManager

- **Classe javax.persistence.EntityManager**
 - Le gestionnaire d'entités est l'interlocuteur principal pour le développeur.
- Les interactions entre la base de données et les beans entité sont assurées par un objet de type javax.persistence.EntityManager : il permet de lire et rechercher des données mais aussi de les mettre à jour (ajout, modification, suppression). L'EntityManager est donc au coeur de toutes les actions de persistance.
 - Il fournit les méthodes pour gérer les entités :
 - les rendre persistantes,
 - les supprimer de la base de données,
 - retrouver leurs valeurs dans la base,
 - etc.

Cycle de vie d'un EntityManager

- La méthode `createEntityManager()` de la classe `EntityManagerFactory` créé un EntityManager
 - L'Entity Manager est supprimé avec la méthode `close()` de la classe `EntityManager`, il ne sera plus possible de l'utiliser ensuite.

Fabrique de l'EntityManager

- La classe Persistence permet d'obtenir une fabrique de gestionnaire d'entités par la méthode **createEntityManagerFactory**
- **2 variantes surchargées de cette méthode :**
 - 1 seul paramètre qui donne le nom de l'unité de persistance (définie dans le fichier **persistence.xml**)
 - Un 2ème paramètre de type Map qui contient des valeurs qui vont écraser les propriétés par défaut contenues dans persistence.xml

Méthodes de EntityManager

- `void flush()`
 - Toutes les modifications effectuées sur les entités du contexte de persistance gérées par l'EntityManager sont enregistrées dans la BD
- `void persist(Object entité)`
 - Une entité nouvelle devient une entité gérée, l'état de l'entité sera sauvegardé dans la BD au prochain *flush ou commit*.
- `void remove(Object entité)`
 - Une entité gérée devient supprimée, les données correspondantes seront supprimées de la BD

Méthodes de EntityManager

- `void lock(Object entité, LockModeType lockMode)`
 - Le fournisseur de persistance gère les accès concurrents aux données de la BD représentées par les entités avec une stratégie optimiste, lock permet de modifier la manière de gérer les accès concurrents à une entité
- `void refresh(Object entité)`
 - L'EntityManager peut synchroniser avec la BD une entité qu'il gère en rafraîchissant son état en mémoire avec les données actuellement dans la BD.
 - Utiliser cette méthode pour s'assurer que l'entité a les mêmes données que la BD.
- `<T> T find(Class<T> classeEntité, Object cléPrimaire)`
 - La recherche est polymorphe : l'entité récupérée peut être de la classe passée en paramètre ou d'une sous-classe (renvoie **null si aucune entité n'a** l'identificateur passé en paramètre)

Utilisation de EntityManager pour la création d'une occurrence

- Pour insérer une nouvelle entité dans la base de données, il faut :
 - Instancier une occurrence de la classe de l'entité
 - Initialiser les propriétés de l'entité
 - Définir les relations de l'entité avec d'autres entités, et au besoin
 - Utiliser la méthode persist() de l'EntityManager en passant en paramètre l'entité

EXEMPLE

```
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;

public class TestJPA {

    public static void main(String[] args) {
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("MaBaseDeTestPU");
        EntityManager em = emf.createEntityManager();
        EntityTransaction transac = em.getTransaction();
        transac.begin();
        Personne nouvellePersonne = new Personne();
        nouvellePersonne.setId(4);
        nouvellePersonne.setNom("nom4");
        nouvellePersonne.setPrenom("prenom4");
        em.persist(nouvellePersonne);
        transac.commit();
        em.close();
        emf.close();
    }
}
```

Utilisation de EntityManager pour rechercher des occurrences

- Pour effectuer des recherches de données, l'EntityManager propose deux mécanismes :
 - La recherche à partir de la clé primaire
 - La recherche à partir d'une requête utilisant une syntaxe dédiée
- Pour la recherche par clé primaire, la classe EntityManager possède les méthodes `find()` et `getReference()` qui attendent toutes les deux en paramètres un objet de type `Class` représentant la classe de l'entité et un objet qui contient la valeur de la clé primaire.

Avec Find()

```
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

public class TestJPA5 {

    public static void main(String[] argv) {

        EntityManagerFactory emf = Persistence.createEntityManagerFactory("MaBaseDeTestPU");
        EntityManager em = emf.createEntityManager(); Personne personne = em.find(Personne.class, 4);

        if (personne != null) {
            System.out.println("Personne.nom=" + personne.getNom());
        }

        em.close();
        emf.close();
    }
}
```

Avec getReference()

```
● ● ●

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityNotFoundException;
import javax.persistence.Persistence;

public class TestJPA6 {

    public static void main(String[] argv) {

        EntityManagerFactory emf = Persistence.createEntityManagerFactory("MaBaseDeTestPU");
        EntityManager em = emf.createEntityManager();
        try {
            Personne personne = em.getReference(Personne.class, 5);
            System.out.println("Personne.nom=" + personne.getNom());
        } catch (EntityNotFoundException e) {
            System.out.println("personne non trouvée");
        }
        em.close();
        emf.close();
    }
}
```

Queries

- Il est possible de rechercher des données sur des critères plus complexes que la simple identité.
Les étapes sont alors les suivantes :
 1. Décrire ce qui est recherché (langage JPQL)
 2. Créer une instance de type **Query**
 3. Initialiser la requête (paramètres, pagination)
 4. Lancer l'exécution de la requête
- JPA introduit le **JPQL (*Java Persistence Query Language*)**, qui est, tout comme le EJBQL ou encore le HQL, un langage de requête du modèle objet, basé sur SQL.

Interface Query

- **Représente une requête**
 - Une instance de **Query** (d'une classe **implémentant Query**) est obtenue des méthodes dédiées de la classe EntityManager :
 - **createQuery ()**,
 - **createNativeQuery ()** ou
 - **createNamedQuery ()**

L'objet Query

```
● ● ●

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import javax.persistence.Query;

public class TestJPA7 {

    public static void main(String[] argv) {

        EntityManagerFactory emf = Persistence.createEntityManagerFactory("MaBaseDeTestPU");
        EntityManager em = emf.createEntityManager();
        Query query = em.createQuery("select p from Personne p where p.nom='nom2'");
        Personne personne = (Personne) query.getSingleResult();

        if (personne == null) {
            System.out.println("Personne non trouvée");
        } else {
            System.out.println("Personne.nom=" + personne.getNom());
        }
        em.close();
        emf.close();
    }
}
```

Query+la méthode getResultList()

```
● ● ●

import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import javax.persistence.Query;

public class TestJPA8 {

    public static void main(String[] argv) {

        EntityManagerFactory emf = Persistence.createEntityManagerFactory("MaBaseDeTestPU");
        EntityManager em = emf.createEntityManager();
        Query query = em.createQuery("select p.nom from Personne p where p.id > 2");
        List noms = query.getResultList();
        for (Object nom : noms) {
            System.out.println("nom = "+nom);
        }
        em.close();
        emf.close();
    }
}
```

Les paramètres nommés+ `setParameter()`

```
import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import javax.persistence.Query;

public class TestJPA9 {

    public static void main(String[] argv) {

        EntityManagerFactory emf = Persistence.createEntityManagerFactory("MaBaseDeTestPU");
        EntityManager em = emf.createEntityManager();
        Query query = em.createQuery("select p.nom from Personne p where p.id > :id");

        query.setParameter("id", 1);

        List noms = query.getResultList();

        for (Object nom : noms) {
            System.out.println("nom = "+nom);
        }

        em.close();

        emf.close();
    }
}
```

Modifier une occurrence avec le EntityManager

- Pour modifier une entité existante dans la base de données, il faut :
 - Obtenir une instance de l'entité à modifier (par recherche sur la clé primaire ou l'exécution d'une requête)
 - Modifier les propriétés de l'entité
 - Selon le mode de synchronisation des données de l'EntityManager, il peut être nécessaire d'appeler la méthode `flush()` explicitement

Exemple

```
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;
import javax.persistence.Query;

public class TestJPA10 {

    public static void main(String[] argv) {

        EntityManagerFactory emf = Persistence.createEntityManagerFactory("MaBaseDeTestPU");
        EntityManager em = emf.createEntityManager();
        EntityTransaction transac = em.getTransaction();

        transac.begin();
        Query query = em.createQuery("select p from Personne p where p.nom='nom2'");
        Personne personne = (Personne) query.getSingleResult();
        if (personne == null) {
            System.out.println("Personne non trouvée");
        }else {
            System.out.println("Personne.prenom=" + personne.getPrenom());
            personne.setPrenom("prenom2 modifié");

            em.flush();
            personne = (Personne) query.getSingleResult();
            System.out.println("Personne.prenom=" + personne.getPrenom());
        }

        transac.commit();
    }
}
```

Supprimer une occurrence avec EntityManager

- Pour supprimer une entité existante dans la base de données:
 - Obtenir une instance de l'entité à supprimer (par recherche sur la clé primaire ou l'exécution d'une requête)
 - Appeler la méthode `remove()` de l'EntityManager en lui passant en paramètre l'instance de l'entité

Exemple

```
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;

public class TestJPA12 {

    public static void main(String[] argv) {

        EntityManagerFactory emf = Persistence.createEntityManagerFactory( "MaBaseDeTestPU" );
        EntityManager em = emf.createEntityManager();  EntityTransaction transac = em.getTransaction();
        transac.begin();

        Personne personne = em.find(Personne.class, 4);

        if (personne == null) {
            System.out.println("Personne non trouvée");
        } else {
            em.remove(personne);
        }
        transac.commit();
        em.close();
        emf.close();
    }
}
```

Rafraîchir les données d'une occurrence

```
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;

public class TestJPA13 {

    public static void main(String[] argv) {
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("MaBaseDeTestPU");
        EntityManager em = emf.createEntityManager();
        EntityTransaction transac = em.getTransaction();
        transac.begin();
        Personne personne = em.find(Personne.class, 4);

        if (personne == null) {
            System.out.println("Personne non trouvée");
        } else {
            em.refresh(personne);
        }
        transac.commit();
        em.close();
        emf.close();
    }
}
```

Queries: NamedQueries

- Peut être mise dans n'importe quelle entité, mais on choisira le plus souvent l'entité qui correspond à ce qui est renvoyé par la requête
- On peut sauvegarder des gabarits de requête dans nos entités. Ceci permet :
 - La réutilisation de la requête
 - D'externaliser les requête du code.

Queries: NativeQueries

- **Une façon de faire des requête en SQL natif.**
 - Sert principalement à avoir plus de contrôle sur les requêtes à la base de donnée

```
public class MyService {  
    public void myMethod() {  
        ...  
        List results = em.createNativeQuery("SELECT * FROM MyPojo", MyPojo.class).getResultList();  
        ...  
    }  
}
```