JPQL : Java Persistence Query Language

requête JPQL :

```
select item from Item item where item.wording = 'descItem'
from Item as item where item.wording = 'descItem'
from Item where wording = 'descItem'
```

En JPQL on sélectionne des objets d'une « Entity » (une classe annotée par @Entity).
Rappelons que le nom d'une entité est donné par l'attribut name de l'annotation @Entity - par exemple, @Entity (name = "EntityItem ") - . S'il n'est pas précisé, alors le nom par défaut de l'entité est le nom complet de la classe.

L'exécution d'une requête passe par la création d'un objet de type Query à partir de l'*entity manager*. Il faut donc un objet de la classe `entityManager` (nommé par la suite `em`) :

```
EntityManager em = ... ;
```

Exécuter une requête JPQL se fait donc en deux temps :

- définition de l'objet requête proprement dit, de type `Query`.
- exécution de la requête, et récupération du résultat.

Notons que la classe `Query` possède également la méthode :

- `getSingleResult()` : retourne l'unique objet résultat de cette requête. Si la requête retourne plusieurs objets, alors une exception de type `NonUniqueResultException` est générée.

Usage :

```
 // construction d'un objet Query
Query query = em.createQuery(
  "select item from Item item where item.wording =  'descItem' ") ;

// exécution et récupération de la liste résultat
List<Item> items = query.getResultList() ;

// analyse du résultat (classique)
for (Item item : items) {   System.out.println("   "+item) ;  }
```

Usage :

```
// construction d'un objet Query
Query query = em.createQuery(
   " from Item as item where item.wording =  'descItem' ") ;

// exécution et récupération de la liste résultat
List<Item> items = query.getResultList() ;

// analyse du résultat (classique)
for (Item item : items) {   System.out.println("   "+item) ;  }
```

Usage :

```
// construction d'un objet Query
Query query = em.createQuery(
   " from Item where wording =  'descItem' ") ;

// exécution et récupération de la liste résultat
List<Item> items = query.getResultList() ;

for (Item item : items) {   System.out.println("   "+item) ;  }
```

requête JPQL paramétrée

```
select item from Item item where item.wording = :valeur
from Item as item where item.wording = :valeur
from Item where wording = :valeur
```

`:valeur` est le paramètre qui sera instancié via la méthode `setParameter` de `Query`.

Usage :

```
Query query = em.createQuery(
    "select item from Item item where item.wording = :valeur") ;

query = query.setParameter("valeur", "descItem");

List<Item> items = query.getResultList() ;
```

Méta-Modèle

Les classes du méta-modèle et ses attributs sont utilisés dans les requêtes pour ne pas avoir à utiliser les noms des tables et champs de la Base de données. Les classes du méta-modèle sont construites par Maven.
Elles sont donc localisées dans le répertoire : `target/generated-sources/apt` du projet.

Exemple :

```
@StaticMetamodel(Item.class)
public abstract class Item_ {
   public static volatile SingularAttribute<Item, String> wording;
   public static volatile SingularAttribute<Item, Integer> id;
}

@StaticMetamodel(ItemPrice.class)
public abstract class Item_ {
   public static volatile SingularAttribute<ItemPrice, Float> price;
   public static volatile SingularAttribute<ItemPrice, Item> item;
   public static volatile SingularAttribute<ItemPrice, Integer> id;
}


@StaticMetamodel(Shop.class)
public abstract class Shop_ {
   public static volatile SingularAttribute<Shop, String> address;
   public static volatile SingularAttribute<Shop, String> name;
   public static volatile SingularAttribute<Shop, Integer> id;
   public static volatile CollectionAttribute<Shop, ItemPrice> catalog;
}
```

Usage du méta-modèle en vue d'éviter d'utiliser les noms des tables et des colonnes de la BD :
`select item from Item item where item.wording = 'descItem'`

```
String queryS = new String(" select item from ");
queryS = queryS.concat(Item.class.getName());
queryS = queryS.concat(" item where item.");
queryS = queryS.concat(Item_.wording.getName());
queryS = queryS.concat(" = 'descItem' ");

Query query = em.createQuery(queryS);
List<Article> articles =  query.getResultList() ;
```

Usage de stringBuilder : `from Item where wording = 'descItem'`

```
StringBuilder querySB = new StringBuilder(" from ");
querySB.append(Item.class.getName());
querySB.append(" where ");
querySB.append(Item_.wording.getName());
querySB.append(" = 'descItem' ");

Query query  = em.createQuery(querySB.toString());

List<Item> items =  query.getResultList() ;
```

Usage dans le cadre d'une requête paramétrée : `from Item where wording = :valeur`

```
StringBuilder querySB = new StringBuilder(" from ");
querySB.append(Item.class.getName());
querySB.append(" where ");
querySB.append(Item_.wording.getName());
querySB.append(" = :valeur");

Query query  = em.createQuery(querySB.toString());
query.setParameter("valeur", "descItem");

List<Item> items =  query.getResultList() ;
```

Usage dans le cadre d'une requête paramétrée par un objet :
`from ItemPrice where item = :valeur`

```
StringBuilder querySB = new StringBuilder(" from ");
querySB.append(ItemPrice.class.getName());
querySB.append("  where ");
querySB.append(ItemPrice_.item.getName());
querySB.append(" = :valeur");
String queryS querySB.toString();

Query query = em.createQuery(queryS);
query.setParameter("valeur", a); // a est un objet de la classe
Item

List<ItemPrice> prices = query.getResultList();
```