

# Criteria API

---

adriencaubel.fr

# Table des matières

- 1. Introduction
  - 1. Rappels
  - 2. Objectifs
  - 3. Exemple
  - 4. Pourquoi l'utiliser
- 2. Criteria Builder
  - 1. Qu'est-ce que CriteriaBuilder ?
  - 2. Quelques méthodes
  - 3. Exemple complet
- 3. Criteria Query
  - 1. Qu'est-ce que CriteriaQuery ?
  - 2. Les objets de base
  - 3. Exemple complet
  - 4. Comment obtenir un CriteriaQuery ?
  - 5. Quelques méthodes
  - 6. 1. Définir la racine : from

# Introduction

# Rappels

- Nous avons étudié les concepts suivants
  - La notion `@Entity`
  - Les associations `@OneToMany` ... et `mappedBy` (bidirectionnelle)
  - L'héritage `@Inheritance`
  - L'optimisation des lectures ( `Fetch.LAZY` / `Fetch.EAGER` )

Nous allons maintenant nous intéresser à **l'API Criteria**

# Objectifs

The basic semantics of a Criteria query consists of a SELECT clause, a FROM clause, and an optional WHERE clause, similar to a JPQL query. Criteria queries set these clauses by using Java programming language objects, so the query can be created in a typesafe manner.

- Une API pour écrire des requête SQL directement avec des méthodes Java

# Example

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("ma-pu");
EntityManager em = emf.createEntityManager();

CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Pet> cq = cb.createQuery(Pet.class);
Root<Pet> pet = cq.from(Pet.class);           // from
cq.select(pet);                             // select
TypedQuery<Pet> q = em.createQuery(cq);
List<Pet> allPets = q.getResultList();
```

<=>

```
List<Pet> = em.createQuery("SELECT p FROM Pet p", Pet.class).getResultList();
```

# Pourquoi l'utiliser

Jusqu'à présent, les opérations de base sur les entités (persist, merge, remove) ne permettaient pas de contrôler les conditions WHERE lors de leur exécution. Plusieurs solutions sont possibles :

- faire un filtre directement en Java
- utiliser du JPQL + `createQuery()`
- utiliser l'API Criteria

# Criteria Builder



# Qu'est-ce que CriteriaBuilder ?

`CriteriaBuilder` est l'**objet principal** qui fournit toutes les méthodes nécessaires pour construire :

- des requêtes ( `CriteriaQuery` )
- des conditions ( `Predicate` )
- des expressions ( `Expression` )
- des fonctions ( `count` , `sum` , `max` , etc.)
- des opérations logiques ( `and` , `or` , `not` )
- des tris ( `orderBy` )

## Comment obtenir un `CriteriaBuilder`

Le `CriteriaBuilder` est fourni par l' `EntityManager` .

```
EntityManager em = ...;  
  
CriteriaBuilder cb = em.getCriteriaBuilder();
```

# Quelques méthodes

L'ensemble des méthodes est disponible dans la [Javadoc](#). On remarque de type principaux : `Expression` et `Predicate`

Méthode	JavaDoc officielle	Signature
<code>equal</code>	Crée un prédicat pour tester l'égalité entre deux expressions.	<code>Predicate equal(Expression&lt;?&gt; x, Object y)</code>
<code>greaterThan</code>	Tester si une expression est strictement supérieure à une autre.	<code>Comparable&lt;? super Y&gt; Predicate gt(Expression&lt;? extends Y&gt; x, Y y)</code>
<code>and</code>	Crée une conjonction de prédicats. Un prédicat AND est vrai si tous les prédicats le sont.	<code>Predicate and(Predicate... restrictions)</code>

# Comprendre Expression

Une `Expression<T>` représente une valeur dans une requête.

Cela peut être :

- un champ (name, age)
- une fonction (count, sum)
- un calcul (age + 10)
- une valeur constante

```
// accéder à un attribut; root.get("name") retourne une Expression<String>  
Expression<String> nameExp = root.get("name");
```

```
// fonction SQL COUNT  
Expression<Long> countExp = cb.count(root);
```

# Comprendre Predicate

Un Predicate représente une condition logique ( WHERE ). Il est toujours construit via CriteriaBuilder

- Predicate equal(Expression<?> x, Object y)

```
// WHERE name = 'Adrien'  
Predicate condition = cb.equal(root.get("name"), "Adrien");
```

```
// WHERE age > 18  
Predicate condition = cb.greaterThan(root.get("age"), 18);
```

On peut également combiner plusieurs prédicat via `and(Predicate... restrictions)`

```
Predicate p1 = cb.equal(root.get("active"), true);  
Predicate p2 = cb.greaterThan(root.get("age"), 18);  
  
Predicate andCondition = cb.and(p1, p2);
```

# Exemple complet

Pour le moment, nous n'avons pas vu comment construire ni l'objet CriteriaQuery, ni l'objet Root

```
CriteriaBuilder cb = em.getCriteriaBuilder(); // 1. Créer un objet CriteriaBuilder
CriteriaQuery<User> query = cb.createQuery(User.class);

Root<User> root = query.from(User.class);

Predicate condition = cb.equal(root.get("name"), "Adrien"); // 2. créer un prédicat

query.select(root).where(condition); // 3. exploiter un prédicat

List<User> results = em.createQuery(query).getResultList();
```

# Criteria Query

# Qu'est-ce que CriteriaQuery ?

CriteriaQuery est une interface qui permet de construire une requête typée. Elle représente donc **la requête elle-même**, c'est-à-dire la structure globale :

- **Définition de la source** ( `.from()` ) : elle renvoie un objet `Root<T>` , qui représente cette entité dans la requête et permet d'accéder aux attributs de celle-ci.
- **Les sélections** ( `.select()` ) : pour définir les champs à récupérer.
- **Les filtres** ( `.where()` ) : pour appliquer des conditions de filtrage.
- **Les jointures** ( `.join()` ) : pour lier plusieurs entités.
- **Les tris** ( `.orderBy()` ) : pour ordonner les résultats.
- **Les agrégations** ( `.groupBy()` , `.having()` ) : pour grouper et appliquer des conditions sur les groupes.

## 3 Types d'opérations

- `CriteriaQuery` pour le SELECT.
- `CriteriaUpdate` pour l'UPDATE.
- `CriteriaDelete` pour le DELETE.

# Les objets de base

## L'interface `Root`

L'interface `Root` représente les entités racines impliquées dans la requête. Il peut y avoir plusieurs racines définies dans une même requête `Criteria`.

=> c'est la clause `FROM`

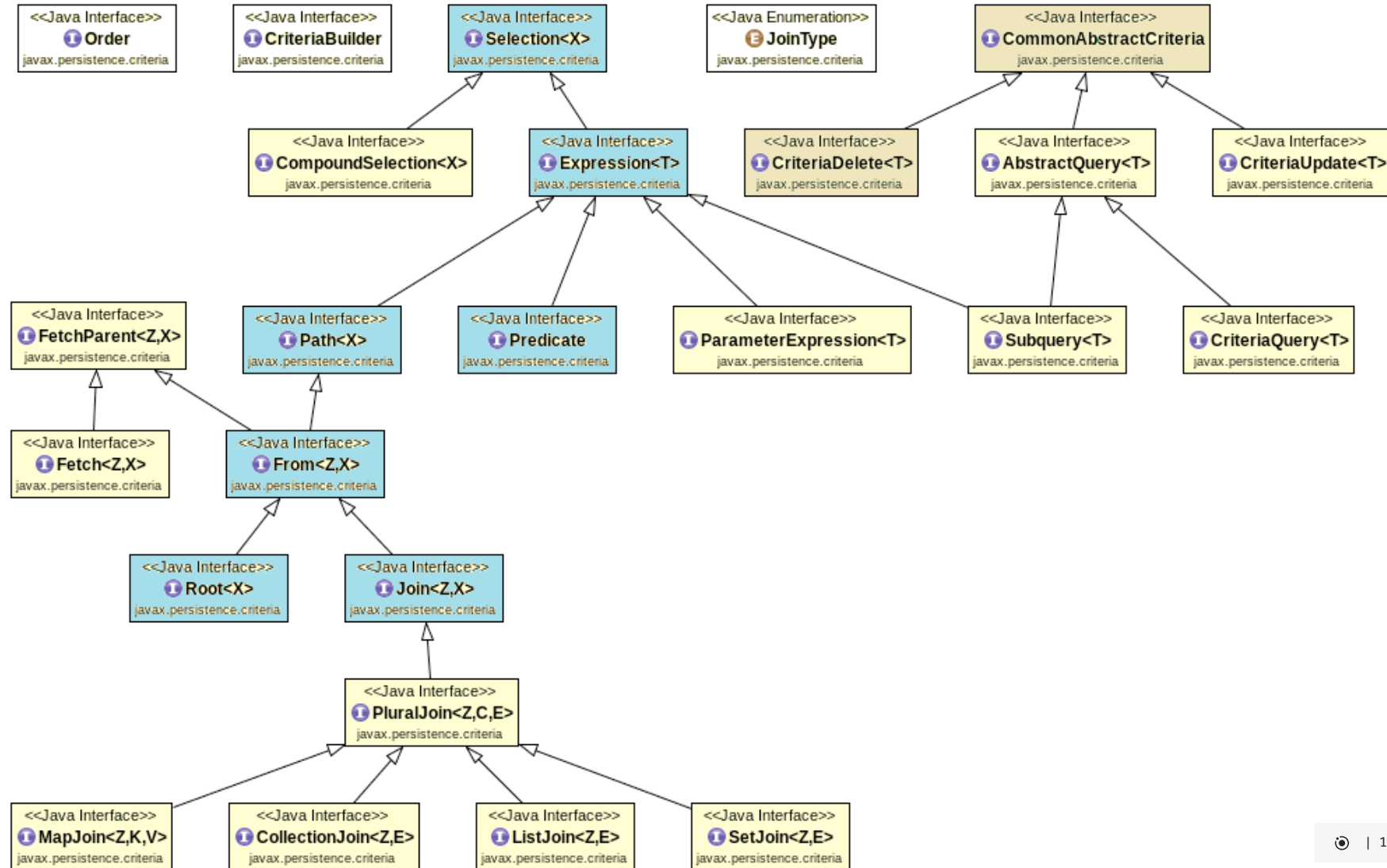
## L'interface `Path`

L'interface `Path` représente le chemin vers un attribut dans l'entité racine. Elle étend également l'interface `Expression`, qui contient des méthodes retournant des `Predicate`.

## La classe `TypedQuery`

- Une instance de `TypedQuery` est nécessaire pour exécuter la `CriteriaQuery`.





Dans le diagramme précédent explique la hiérarchie d'héritage entre les différentes interfaces présentes dans l'API Criteria.

- `Selection` se trouve au sommet et est étendue par `Expression`.
  - `Expression` est à son tour étendue par les interfaces `Predicate` et `Path`.
  - `From` étant `Path` qui est à son tour le parent des interfaces `Root` et `Join`.
- L'interface `Root` est également une expression.
  - Cela signifie que nous pouvons interroger une entité complète en passant `Root` comme paramètre à la méthode `select`.
  - Si nous voulons récupérer un attribut sélectionné, nous pouvons récupérer le chemin d'accès à l'attribut à l'aide de `root.getAttributeName()`. Cette méthode renvoie un objet `Path` qui hérite de `Expression`.

# Exemple complet

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("MaBaseDeTestPU");
EntityManager em = emf.createEntityManager();

CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Pet> cq = cb.createQuery(Pet.class);
Root<Pet> pet = cq.from(Pet.class);
cq.select(pet);
TypedQuery<Pet> q = em.createQuery(cq);
List<Pet> allPets = q.getResultList();
```

1. Cette requête illustre les étapes de base pour créer une requête Criteria.
2. Utiliser une instance de `EntityManager` pour créer un objet `CriteriaBuilder`.
3. Créer un objet requête en instanciant l'interface `CriteriaQuery`.  
Les attributs de cet objet requête seront ensuite modifiés avec les détails de la requête.
4. Définir la racine de la requête en appelant la méthode `from` sur l'objet `CriteriaQuery`.
5. Spécifier le type du résultat de la requête en appelant la méthode `select` de l'objet `CriteriaQuery`.
6. Préparer la requête pour son exécution en créant une instance de `TypedQuery<T>`.
7. Exécuter la requête en appelant la méthode `getResultList` sur l'objet `TypedQuery<T>`.  
Comme cette requête retourne une collection d'entités, le résultat est stocké dans une `List`.

# Comment obtenir un CriteriaQuery ?

Depuis un `CriteriaBuilder` on obtient une instance typé Une requête Criteria suit toujours ce schéma :

- `CriteriaBuilder`
- `CriteriaQuery`
- `Root`
- `select`
- `where`
- `TypedQuery`

```
CriteriaBuilder cb = em.getCriteriaBuilder();

CriteriaQuery<User> query = cb.createQuery(User.class);

// Puis obtenir la root, faire un select et récupérer la réponse
```

# Quelques méthodes

L'ensemble des méthodes est disponible dans la [Javadoc](#). Ci-dessous nous en présentons quelques une.

Méthode	JavaDoc officielle	Signature
<code>from()</code>	Crée et ajoute une racine de requête correspondant à l'entité donnée, formant ainsi la clause <code>FROM</code> .	<code>&lt;X&gt; Root&lt;X&gt; from(Class&lt;X&gt; entityClass)</code>
<code>select()</code>	Spécifie l'élément à retourner dans le résultat de la requête.	<code>CriteriaQuery&lt;T&gt; select(Selection&lt;? extends T&gt; selection)</code>
<code>where()</code>	Modifie la requête pour appliquer les restrictions données.	<code>CriteriaQuery&lt;T&gt; where(Expression&lt;Boolean&gt; restriction)</code>
<code>groupBy()</code>	Spécifie les expressions utilisées pour regrouper les résultats de la requête.	<code>CriteriaQuery&lt;T&gt; groupBy(Expression&lt;? &gt;... grouping)</code>

# 1. Définir la racine : from

```
// Equivalent SQL : FROM User u  
Root<User> root = query.from(User.class);
```

Soit nous exploitons directement l'objet `root` dans un select, par exemple `query.select(root);`. Soit nous pouvons créer une jointure

## Jointures : join

```
// Join<Source, Destination>  
Join<User, Department> dept = root.join("department");  
  
query.where(cb.equal(dept.get("name"), "IT"));
```

## 2. Sélectionner des résultats : select

```
// Equivalent SQL : SELECT u  
query.select(root);
```

### Sélectionner un champ précis

Si on souhaite récupérer un champ précis, nous devons passer un objet de type `Expression`, ici `root.get("name")`

```
query.select(root.get("name"));
```

### 3. Exécution avec TypedQuery

Une fois la requête terminée, l'exécuté et récupérer le résultat

```
TypedQuery<User> typedQuery = em.createQuery(query);  
  
List<User> results = typedQuery.getResultList();
```



# Exemple complet

```
// Créer un objet de type CriteriaQuery
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Pet> cq = cb.createQuery(Pet.class);

// Construire la requête
Root<Pet> pet = cq.from(Pet.class);
Join<Pet, Owner> owner = pet.join("owner"); // Jointure

cq.select(pet)
  .where(
    cb.equal(pet.get("type"), "dog"),
    cb.lessThan(pet.get("age"), 5),
    cb.equal(owner.get("id"), ownerId)
  )
  .orderBy(cb.asc(pet.get("name")));

// Executer la requête
TypedQuery<Pet> q = em.createQuery(cq);
List<Pet> filteredPets = q.getResultList();
```