Guide des Critères JPA - Stilles Auto

📚 Introduction aux Critères JPA

Les critères JPA offrent une API type-safe pour construire des requêtes dynamiques sans utiliser JPQL ou SQL natif.

🏗️ Structure de Base

Composants Principaux

```
// 1. CriteriaBuilder - Construit les critères
CriteriaBuilder cb = entityManager.getCriteriaBuilder();

// 2. CriteriaQuery - Définit la requête
CriteriaQuery<Vehicle> query = cb.createQuery(Vehicle.class);

// 3. Root - Entité racine
Root<Vehicle> root = query.from(Vehicle.class);

// 4. Prédicats - Conditions
Predicate predicate = cb.equal(root.get("brand"), "Toyota");

// 5. Exécution
query.where(predicate);
TypedQuery<Vehicle> typedQuery = entityManager.createQuery(query);
List<Vehicle> results = typedQuery.getResultList();
```

🔍 Exemples de Critères

1. Recherche Simple (Equal)

```
// Trouver les véhicules par marque
CriteriaBuilder cb = entityManager.getCriteriaBuilder();
CriteriaQuery<Vehicle> query = cb.createQuery(Vehicle.class);
Root<Vehicle> root = query.from(Vehicle.class);

Predicate brandPredicate = cb.equal(root.get("brand"), "Toyota");
query.where(brandPredicate);

return entityManager.createQuery(query).getResultList();
```

2. Recherche Like (Contient)

```

```

```
// Rechercher les véhicules contenant "Corolla"
Predicate modelPredicate = cb.like(
    root.get("model"),
    "%" + searchTerm + "%"
);
query.where(modelPredicate);
```

3. Recherche Booléenne (True/False)

```
// Trouver les véhicules disponibles
Predicate availablePredicate = cb.equal(root.get("available"), true);
query.where(availablePredicate);
```

4. Recherche avec OR

```
// Rechercher par marque OU modèle
Predicate brandPredicate = cb.like(root.get("brand"), "%" + term + "%");
Predicate modelPredicate = cb.like(root.get("model"), "%" + term + "%");
Predicate orPredicate = cb.or(brandPredicate, modelPredicate);
query.where(orPredicate);
```

5. Recherche avec AND

```
// Véhicules disponibles ET de marque Toyota
Predicate availablePredicate = cb.equal(root.get("available"), true);
Predicate brandPredicate = cb.equal(root.get("brand"), "Toyota");
Predicate andPredicate = cb.and(availablePredicate, brandPredicate);
query.where(andPredicate);
```

6. Recherche avec IN

```
// Véhicules avec statut AVAILABLE ou MAINTENANCE
List<VehicleStatus> statuses = Arrays.asList(
    VehicleStatus.AVAILABLE,
    VehicleStatus.MAINTENANCE
);
Predicate statusPredicate = root.get("status").in(statuses);
query.where(statusPredicate);
```

7. Recherche avec Comparaison Numérique

```
// Véhicules avec prix de location > 50
Predicate pricePredicate = cb.greaterThan(
    root.get("dailyRentalPrice"),
    new BigDecimal("50")
);
query.where(pricePredicate);
```

8. Tri (Order By)

```
// Trier par marque ascendant
query.orderBy(cb.asc(root.get("brand")));

// Trier par prix descendant
query.orderBy(cb.desc(root.get("dailyRentalPrice")));
```

9. Pagination

```
TypedQuery<Vehicle> typedQuery = entityManager.createQuery(query);
typedQuery.setFirstResult(0);      // Offset
typedQuery.setMaxResults(10);      // Limit
List<Vehicle> results = typedQuery.getResultList();
```

10. Count

```
CriteriaQuery<Long> countQuery = cb.createQuery(Long.class);
Root<Vehicle> countRoot = countQuery.from(Vehicle.class);
countQuery.select(cb.count(countRoot));
countQuery.where(predicate);
Long count = entityManager.createQuery(countQuery).getSingleResult();
```

🔗 Recherche avec Jointures

Join Simple

```
// Trouver les locations d'un client spécifique
CriteriaQuery<Rental> query = cb.createQuery(Rental.class);
Root<Rental> root = query.from(Rental.class);
Join<Rental, User> userJoin = root.join("client");

Predicate clientPredicate = cb.equal(userJoin.get("id"), clientId);
query.where(clientPredicate);
```

Left Join

```
// Inclure les locations même si pas de client
Join<Rental, User> userJoin = root.join("client", JoinType.LEFT);
```

🎯 Implémentation dans le Projet

Pattern pour VehicleRepositoryImpl

```
@Component
public class VehicleRepositoryImpl implements VehicleRepositoryApi {
    private final EntityManager entityManager;

    public Flux<Vehicle> findByBrand(String brand) {
        return Mono.fromCallable(() -> {
            CriteriaBuilder cb = entityManager.getCriteriaBuilder();
            CriteriaQuery<Vehicle> query = cb.createQuery(Vehicle.class);
            Root<Vehicle> root = query.from(Vehicle.class);

            query.where(cb.equal(root.get("brand"), brand));

            return entityManager.createQuery(query).getResultList();
        }).flatMapMany(Flux::fromIterable);
    }

    public Flux<Vehicle> searchByBrandOrModel(String searchTerm) {
        return Mono.fromCallable(() -> {
            CriteriaBuilder cb = entityManager.getCriteriaBuilder();
            CriteriaQuery<Vehicle> query = cb.createQuery(Vehicle.class);
            Root<Vehicle> root = query.from(Vehicle.class);

            String likePattern = "%" + searchTerm + "%";
            Predicate brandPredicate = cb.like(root.get("brand"),
likePattern);
            Predicate modelPredicate = cb.like(root.get("model"),
likePattern);

            query.where(cb.or(brandPredicate, modelPredicate));

            return entityManager.createQuery(query).getResultList();
        }).flatMapMany(Flux::fromIterable);
    }
}
```

⚠ Pièges Courants

1. Oublier le Where

```
// ✗ Mauvais - Retourne tous les véhicules
CriteriaQuery<Vehicle> query = cb.createQuery(Vehicle.class);
Root<Vehicle> root = query.from(Vehicle.class);
// Pas de where!

// ✓ Bon
query.where(predicate);
```

2. Typage Incorrect

```
// ✗ Mauvais - Erreur de type
root.get("dailyRentalPrice").in(Arrays.asList("50", "100"));

// ✓ Bon
root.get("dailyRentalPrice").in(Arrays.asList(
    new BigDecimal("50"),
    new BigDecimal("100")
));
```

3. Null Checks

```
// ✗ Mauvais - Peut causer NPE
if (brand != null) {
    query.where(cb.equal(root.get("brand"), brand));
}

// ✓ Bon - Utiliser isNull
Predicate brandPredicate = brand != null
    ? cb.equal(root.get("brand"), brand)
    : cb.isNotNull(root.get("brand"));
```

📊 Comparaison : Critères vs JPQL

Critères (Type-Safe)

```
CriteriaBuilder cb = entityManager.getCriteriaBuilder();
CriteriaQuery<Vehicle> query = cb.createQuery(Vehicle.class);
Root<Vehicle> root = query.from(Vehicle.class);
query.where(cb.equal(root.get("brand"), "Toyota"));
```

JPQL (String-Based)

```

```

```
String jpql = "SELECT v FROM Vehicle v WHERE v.brand = :brand";
TypedQuery<Vehicle> query = entityManager.createQuery(jpql, Vehicle.class);
query.setParameter("brand", "Toyota");
```

Avantages des Critères

- ✅ Type-safe
- ✅ Refactoring-friendly
- ✅ Autocomplétion IDE
- ✅ Erreurs détectées à la compilation

🚀 Bonnes Pratiques

1. **Toujours utiliser les critères** plutôt que JPQL ou SQL natif
2. **Valider les entrées** avant de les utiliser dans les critères
3. **Utiliser des prédicats réutilisables** pour les conditions complexes
4. **Tester les critères** avec différentes valeurs
5. **Documenter les critères complexes** avec des commentaires

📚 Ressources

- JPA Criteria API Documentation
- Spring Data JPA Criteria
- Hibernate Criteria

Dernière mise à jour : 30 Novembre 2025 **Status** : Complet