

L'interrogation des entités JPA avec JPQL

Emmanuel BRUNO

Université de Toulon, LIS UMR CNRS 7020

emmanuel.bruno@univ-tln.fr

2026-10-14

Table des matières

1	Types d'interrogation JPA	1
2	SQL Natif	1
2.1	Mapping vers une liste d'entités	1
2.2	Résultat unique	2
2.3	Résultats multiples non mappés	2
2.4	Pagination	2
2.5	SQL Natif (Aggregation)	3
2.6	SQL Natif (paramètres)	3
3	Java Persistence Query Language (JPQL) .	4
3.1	Exemple de requête JPQL	4
3.2	Éléments de syntaxe JPQL	5
3.3	Exemple de requêtes JPQL	5
3.4	Exemple d'exécution de requêtes JPQL	5
4	Projections JPQL	5
4.1	Exemple de projection sur un DTO .	6
4.2	Agrégations JPQL	6
4.3	Exemple de comptage	6
4.4	Jointures JPQL	6
5	Named Queries	7
6	Criteria API	7
7	Entity Graphs	7
7.1	Named Entity Graph	8
7.2	Entity Graph dynamique	8
8	Performance et Optimisation JPA	8
8.1	Exemple d'optimisation de requêtes .	8
9	Repository Pattern avec JPA	9
9.1	Une interface générique de repository	9

9.2	Implémentation partielle de repository JPA	9
9.3	Repository JPA avec requêtes personnalisées	10
9.4	Utilisation du repository	10
10	Interrogation avec JPA : Points clés . . .	10

1 Types d'interrogation JPA

- Différentes approches disponibles:
 - SQL natif (`createNativeQuery`)
 - JPQL (`createQuery`)
 - Langage de requête orienté objet inspiré de SQL pour les entités
 - Criteria API (`criteriaBuilder`)
 - Constructeur dynamique et typesafe de requêtes
 - Named Queries (`@NamedQuery`, `@NamedNativeQuery`, ...)
 - Requêtes nommées dans les entités.

2 SQL Natif

- Requêtes SQL classiques via `createNativeQuery()`
- Paramétrage possible avec `?` ou `:name`
- Mapping vers entités ou résultats bruts
- Utile pour requêtes complexes/spécifiques

2.1 Mapping vers une liste d'entités

- `getResultList()` retourne une liste d'entités

- Mapping automatique des colonnes vers les attributs

```
try (EntityManager entityManager =
emf.createEntityManager()) {
    Query query =
entityManager.createNativeQuery(
    "SELECT * FROM CUSTOMER WHERE
lastname LIKE ?",
    Customer.class);
    List<Customer> customers = query
    .setParameter(1, "C%")
    .setMaxResults(3)
    .getResultList();

    customers.forEach(c ->
System.out.println("%s, %s (%s)"
    .formatted(c.getLastname(),
c.getFirstname(), c.getEmail())));
}
```

```
Corwin, Ta (Ta.Corwin@koch.name)
Cruikshank, Whitney
(Whitney.Cruikshank@medhurst.biz)
Crooks,
Prince (Prince.Crooks@conroy.biz)
```

2.2 Résultat unique

- `getSingleResult()`

```
try (EntityManager entityManager =
emf.createEntityManager()) {
    String lastname = (String)
entityManager.createNativeQuery(
    "SELECT lastname FROM CUSTOMER
WHERE id = :id")
    .setParameter("id", 1)
    .getSingleResult();
    System.out.println(lastname);
}
```

```
Lockman
```

2.3 Résultats multiples non mapés

```
try (EntityManager entityManager =
emf.createEntityManager()) {
    List<Object[]> results =
entityManager.createNativeQuery(
    "SELECT id, email FROM CUSTOMER")
    .setMaxResults(3)
    .getResultList();
    results.forEach(r ->
System.out.println("%s,
%s".formatted(r[0], r[1])));
}
```

```
1, Lien.Lockman@schuster.io
2, Roseline.Sauer@schultz.org
3, Jacquelyn.Wyman@lind.com
```

2.4 Pagination

- Utilisation de `setFirstResult` et `setMaxResults`

```
try (EntityManager entityManager =
emf.createEntityManager()) {
    Query query =
entityManager.createNativeQuery(
    ""
    SELECT *
    FROM CUSTOMER
    ORDER BY id""
    Customer.class);

    //Customers (page 10 of size 5)
    int pageNumber = 10;
    int pageSize = 5;
    List<Customer> customers = query
    .setFirstResult((pageNumber-1)*pageSize)
    .setMaxResults(pageSize)
    .getResultList();

    //System.out.println(customers);
}
```

```
customers.stream().map(Customer::toString)
}
```

```
Customer(creationDate=2026-10-14T17:23:04.330059,
id=46,
email=Freeman.Schmitt@mohr.net,
firstname=Freeman, lastname=Schmitt,
displayName=null, birthDate=null,
status=LEAD, version=0)
Customer(creationDate=2026-10-14T17:23:04.330059,
id=47, email=Donella.Prohaska@balistreri.com,
firstname=Donella,
lastname=Prohaska, displayName=null,
birthDate=null, status=LEAD,
version=0)
Customer(creationDate=2026-10-14T17:23:04.330059,
id=48,
email=Monty.Quigley@mcdermott.co,
firstname=Monty, lastname=Quigley,
displayName=null, birthDate=null,
status=LEAD, version=0)
Customer(creationDate=2026-10-14T17:23:04.330059,
id=49,
email=Ward.Hirthe@williamson.biz,
firstname=Ward, lastname=Hirthe,
displayName=null, birthDate=null,
status=LEAD, version=0)
Customer(creationDate=2026-10-14T17:23:04.330059,
id=50,
email=Palma.Doyle@bradtke.com,
firstname=Palma, lastname=Doyle,
displayName=null, birthDate=null,
status=LEAD, version=0)
```

2.5 SQL Natif (Aggregation)

```
try (EntityManager entityManager =
emf.createEntityManager()) {
    long result = (Long) entityManager
        .createNativeQuery(
            """
            SELECT COUNT(1)
            FROM CUSTOMER""")
        .getSingleResult();
}
```

```
.getSingleResult();
log.info("Customers count:
%d".formatted(result));
}
```

```
17:23:06.071 [IJava-executor-0] INFO
notebook -- Customers count: 1000
```

```
String name = "Smith";
try (EntityManager entityManager =
emf.createEntityManager()) {
    name = (String)entityManager.createNativeQuery(
        """
        SELECT LASTNAME FROM CUSTOMER
        ORDER BY RANDOM()
        LIMIT 1""").getSingleResult();
}
log.info("Nom d'une personne
aléatoire: {}",name);
```

```
17:23:06.199 [IJava-executor-0] INFO
notebook -- Nom d'une personne
aléatoire: Hessel
```

2.6 SQL Natif (paramètres)

Les requêtes peuvent être paramétrées (? et setParameter(...)).

```
//Variable name (String) contains an
existing name
try (EntityManager entityManager =
emf.createEntityManager()) {
    Query query =
entityManager.createNativeQuery(
        """
        SELECT *
        FROM CUSTOMER
        WHERE LASTNAME = :lastname""",
        Customer.class);

    List<Customer> customers = query
```

```

        .setParameter("lastname", name)
        .setMaxResults(3)
        .getResultList();
    log.info("3 firsts customers whose
name is %s".formatted(name));
    customers.stream().forEach(System.out::println);
}

```

```

17:23:06.293 [IJava-executor-0] INFO
notebook -- 3 firsts customers whose
name is Hessel
Customer(creationDate=2026-10-14T17:23:04.877342,
id=466,
email=Arnulfo.Hessel@stark.net,
firstname=Arnulfo, lastname=Hessel,
displayName=null, birthDate=null,
status=LEAD, version=0)

```

```

17:23:06.395 [IJava-executor-0] INFO
notebook -- 3 firsts customers whose
name is Hessel
Customer(creationDate=2026-10-14T17:23:04.877342,
id=466,
email=Arnulfo.Hessel@stark.net,
firstname=Arnulfo, lastname=Hessel,
displayName=null, birthDate=null,
status=LEAD, version=0)

```

3 Java Persistence Query Language (JPQL)

- Syntaxe orientée objet
- Indépendant de la base de données
- Supporte les paramètres nommés
- Collection des résultats:
 - `getResultList(): List<T>`
 - `getSingleResult(): T`
 - `getResultStream(): Stream<T>`

JPQL Wiki Book

3.1 Exemple de requête JPQL

```

//Variable name (String) contains an
existing name
try (EntityManager entityManager =
emf.createEntityManager()) {
    Query query =
entityManager.createNativeQuery(
        ""
        SELECT *
        FROM CUSTOMER
        WHERE LASTNAME = ?""",
        Customer.class);

    List<Customer> customers = query
        .setParameter(1, name)
        .setMaxResults(3)
        .getResultList();
    log.info("3 firsts customers whose
name is %s".formatted(name));
    customers.stream().forEach(System.out::println);
}

```

```

// Liste d'entités
TypedQuery<Customer> query =
entityManager.createQuery(
    "SELECT c FROM Customer c WHERE
c.id > :minId",
    Customer.class);
List<Customer> customers = query
    .setParameter("minId", 10)
    .getResultList();

```

```

// Résultat unique
Customer customer =
entityManager.createQuery(
    "SELECT c FROM Customer c WHERE
c.lastname = :lastname",
    Customer.class)
    .setParameter("lastname", name)
    .setMaxResults(1)
    .getSingleResult();

```

possibilité de nommer les paramètres:

```

        .getSingleResult();
    log.info("Customer:
%s".formatted(customer.getEmail()));

```

```

17:23:06.789      [IJava-executor-0]
INFO      notebook  --      Customer:
Arnulfo.Hessel@stark.net

```

3.2 Éléments de syntaxe JPQL

- Clauses principales:
 - SELECT, FROM, WHERE
 - GROUP BY, HAVING
 - ORDER BY
- Expressions de chemin:
 - Navigation: `customer.address.city`
 - Collections: `customer.orders`
 - Clés
composites: `employee.id.department`
- Opérateurs:
 - Comparaison: `=`, `>`, `<`, `>=`, `<=`, `<>`, `BETWEEN`, `LIKE`, `IN`
 - Logiques: `AND`, `OR`, `NOT`
 - NULL: `IS NULL`, `IS NOT NULL`
 - Collections: `IS EMPTY`, `MEMBER OF`
- Fonctions:
 - String: `CONCAT`, `SUBSTRING`, `TRIM`, `LOWER`, `UPPER`
 - Agrégation: `COUNT`, `SUM`, `AVG`, `MAX`, `MIN`
 - Date: `CURRENT_DATE`, `CURRENT_TIME`, `CURRENT_TIMESTAMP`

3.3 Exemple de requêtes JPQL

```

// Chemin composé
"SELECT c FROM Customer c WHERE
c.address.country = :country"

// Opérateurs collections
"SELECT c FROM Customer c WHERE
c.orders IS NOT EMPTY"

```

```

// Fonctions
"SELECT CONCAT(c.firstName, ' ',
c.lastName) FROM Customer c"

```

```

// Sous-requêtes
"SELECT c FROM Customer c
WHERE SIZE(c.orders) > (SELECT
AVG(SIZE(c2.orders)) FROM Customer
c2)"

```

3.4 Exemple d'exécution de requêtes JPQL

```

//Variable name (String) contains an
existing name
TypedQuery<Customer> query =
entityManager
    .createQuery("""
        SELECT c
        FROM Customer c
        WHERE
c.lastname = :name""",
        Customer.class);

query.setParameter("name", name)
    .setMaxResults(3)
    .getResultStream()
    .map(c->c.getFirstname()+"
"+c.getLastname())
    .forEach(System.out::println);

```

Arnulfo Hessel

4 Projections JPQL

- Sélection partielle des attributs
- Construction de DTOs
- Agrégations et groupements
- Types de retour:
 - `Object[]` pour projections multiples
 - Type simple pour projection unique
 - DTO avec constructeur

```
// DTO avec constructeur
List<CustomerDTO> dtos =
em.createQuery(
    "SELECT new
com.example.CustomerDTO(c.name,
c.age) " +
    "FROM Customer c",
CustomerDTO.class)
.getResultList();

// Projection multiple
List<Object[]> results =
em.createQuery(
    "SELECT c.name, c.age FROM
Customer c")
.getResultList();
```

4.1 Exemple de projection sur un DTO

```
package fr.univtlb.bruno.demos.jpa.hello;

/**
 * A DTO (Data Transfer Object) for
 * displaying customer information.
 */
public record
CustomerDisplayDTO(String firstname,
String lastname) {
}
```

```
//Variable name (String) contains an
existing name
TypedQuery<CustomerDisplayDTO> query
= entityManager.createQuery("
SELECT new fr.univtlb.bruno.demos.jpa.hello.CustomerDisplayDTO(
(c.firstname,
c.lastname)
FROM Customer c
WHERE
c.lastname = :name",
CustomerDisplayDTO.class);
```

```
query
    .setParameter("name", name)
    .setMaxResults(3)
    .getResultStream()
    .forEach(System.out::println);
```

```
CustomerDisplayDTO[firstname=Arnulfo,
lastname=Hessel]
```

4.2 Agrégations JPQL

- Fonctions disponibles:
 - ▶ COUNT, SUM, AVG, MIN, MAX
 - ▶ GROUP BY, HAVING

```
// Groupement avec having
"SELECT c.country, COUNT(c),
AVG(c.age)
FROM Customer c
GROUP BY c.country
HAVING COUNT(c) > :minCustomers"

// Sous-requêtes
"SELECT c FROM Customer c WHERE
(SELECT COUNT(o) FROM c.orders o)
> :orderCount"
```

4.3 Exemple de comptage

```
long customerCount=(long)entityManager.createQuery("
SELECT COUNT(c)
FROM
Customer c").getSingleResult();

System.out.println("There is %s customers.".formatted(customerCount));
```

```
There is 1000 customers.
```

4.4 Jointures JPQL

- Types de jointures disponibles:
 - ▶ INNER JOIN

- LEFT JOIN
- JOIN FETCH (chargement eager)

```
// Inner join avec fetch
"SELECT c FROM Customer c JOIN FETCH
c.orders o WHERE o.total > :total"

// Left join
"SELECT DISTINCT c FROM Customer c
LEFT JOIN c.orders o WHERE o.status
= :status"
```

5 Named Queries

- Requêtes définies au niveau de l'entité
- Validées au démarrage de l'application
- Réutilisables et maintenables
- Existent pour toutes les formes de requêtes (JPQL, SQL et procédures stockées).

```
@Entity
@NamedQueries({
    @NamedQuery(
        name = "Customer.findByStatus",
        query = "SELECT c FROM Customer
c WHERE c.status = :status"
    ),
    @NamedQuery(
        name =
"Customer.countByCountry",
        query = "SELECT c.country,
COUNT(c) FROM Customer c GROUP BY
c.country"
    )
})
```

6 Criteria API

- API type-safe pour construire des requêtes
- Alternative programmatique à JPQL

```
import jakarta.persistence.criteria.CriteriaBuilder;
import jakarta.persistence.criteria.CriteriaQuery;
import
```

```
jakarta.persistence.criteria.Root;
import jakarta.persistence.criteria.Predicate;

CriteriaBuilder      cb      =
entityManager.getCriteriaBuilder();
CriteriaQuery<Customer> cq    =
cb.createQuery(Customer.class);
Root<Customer>        customer =
cq.from(Customer.class);

// Conditions
Predicate            statusPredicate =
cb.equal(customer.get("status"),
Customer.Status.LEAD);
Predicate            namePredicate =
cb.like(customer.get("lastname"),
"C%");

cq.select(customer)
    .where(cb.and(statusPredicate,
namePredicate));

entityManager.createQuery(cq)
    .setMaxResults(3)
    .getResultStream()
    .map(c->c.getFirstname()+"
"+c.getLastname()+"
("+c.getStatus()+")")
    .forEach(System.out::println);
```

```
Ta Corwin (LEAD)
Whitney Cruickshank (LEAD)
Prince Crooks (LEAD)
```

7 Entity Graphs

- Contrôle fin du chargement des associations
- Types de graphes:
 - @NamedEntityGraph au niveau entité
 - Entity graphs dynamiques

LOAD vs FETCH

LOAD graph: garde les paramètres de chargement par défaut + ceux du graph

- FETCH graph: charge UNIQUEMENT ce qui est spécifié dans le graph

7.1 Named Entity Graph

- Déclaré au niveau de l'entité

```
@Entity
@NamedEntityGraph(
    name =
    "Customer.withOrders", // Nom
    unique du graph
    attributeNodes = { //
    Liste des associations à charger
    @NamedAttributeNode("orders") //
    Charge l'association orders
    }
)
public class Customer {
    @OneToMany(mappedBy = "customer")
    private Set<Order> orders; //
    Association orders vers Order
}

// Utilisation du Named Entity Graph
// Le graph est récupéré par son nom
et appliqué viai un hint
EntityGraph<?> graph = em.getEntityGraph("
List<Customer> customers =
em.createQuery(
    "SELECT c FROM Customer c",
    Customer.class)
    .setHint("jakarta.persistence.fetchgraph")
    graph) // Active le graph
    .getResultList();
```

7.2 Entity Graph dynamique

- Alternative flexible, créé programmatiquement

```
EntityGraph<Customer> graph = em.createEntityGraph();
graph.addAttributeNodes("orders");
Premier niveau
```

```
graph.addSubgraph("orders").addAttributeNodes("items");
Second niveau
```

8 Performance et Optimisation JPA

- Gestion du cache:
 - Cache L1 (EntityManager): automatique, par transaction
 - Cache L2 (SessionFactory): partagé, configurable
 - Query cache: résultats des requêtes
- Stratégies de chargement:
 - LAZY: chargement à la demande
 - EAGER: chargement immédiat
 - JOIN FETCH: optimisation des requêtes
 - Entity Graphs: contrôle fin
- Pagination et batching:
 - setFirstResult/setMaxResults
 - hibernate.jdbc.batch_size
 - Requêtes par lots (bulk operations)

8.1 Exemple d'optimisation de requêtes

```
// filepath: /examples/performance/
// QueryOptimization.java
// Cache de requête
Query query = em.createQuery(jpql)
    .setHint("jakarta.persistence.cache.storeMode",
    CacheStoreMode.USE)
    .setHint("jakarta.persistence.cache.retrieveMode",
    CacheRetrieveMode.USE);

// Pagination optimisée
List<Customer> customers =
em.createQuery(jpql, Customer.class)
    .setFirstResult(offset)
    .setMaxResults(pageSize)
    .setHint("org.hibernate.readOnly",
    true)
    .getResultList();
```

```
// Opération par lots
int updated = em.createQuery(
    "UPDATE Customer c SET c.status
= :status WHERE c.type = :type")
    .setParameter("status", "VIP")
    .setParameter("type", "PREMIUM")
    .executeUpdate();
```

9 Repository Pattern avec JPA

- Principes:
 - Encapsulation de la logique d'accès aux données
 - Interface générique pour les opérations CRUD
 - Isolation de la couche persistance
 - Tests facilités

9.1 Une interface générique de repository

```
public interface
GenericRepository<T, ID> {
    Optional<T> findById(ID id);
    List<T> findAll(int pageNumber,
int pageSize);
    T save(T entity);
    void delete(T entity);
}
```

9.2 Implémentation partielle de repository JPA

```
public abstract class
JpaRepository<T, ID> implements
GenericRepository<T, ID> {
    //A fournir dans le constructeur
    ou autre (cf. CDI ou Spring)
    protected EntityManager em;

    //A fournir ou à déduire
    private final Class<T>
entityClass;
```

```
protected
JpaRepository(Class<T> entityClass,
EntityManager entityManager) {
    this.em = entityManager;
    this.entityClass = entityClass;
}

@Override
public Optional<T> findById(ID
id) {
    return Optional.ofNullable(em.find(entityClass,
id));
}

@Override
public List<T> findAll(int
pageNumber, int pageSize) {
    //Utiliser des named queries
    ou la criteria API
    String jpql = "SELECT e FROM
" + entityClass.getSimpleName() + " e
ORDER BY e.id";
    return em.createQuery(jpql,
entityClass)
        .setFirstResult(pageNumber*pageSize)
        .setMaxResults(pageSize)
        .getResultList();
}

@Override
public T save(T entity) {
    em.getTransaction().begin();
    if (em.contains(entity)) {
        entity = em.merge(entity);
    } else {
        em.persist(entity);
    }
    em.getTransaction().commit();
    return entity;
}

@Override
```

```

    public void delete(T entity) {
        em.getTransaction().begin();
        em.remove(entity);
        em.getTransaction().commit();
    }
}

```

9.3 Repository JPA avec requêtes personnalisées

```

public class CustomerRepository
extends JpaRepository<Customer,
Long> {

    public
CustomerRepository(EntityManager
entityManager) {
        super(Customer.class,
entityManager);

        public List<Customer>
findByStatus(int pageNumber, int
pageSize, Customer.Status status) {
        return em.createQuery(
            "SELECT c FROM Customer
c WHERE c.status = :status ORDER BY
c.id",
            Customer.class)
            .setParameter("status",
status)
            .setFirstResult(pageNumber*pageSize)
            .setMaxResults(pageSize)
            .getResultList();
        }
    }
}

```

9.4 Utilisation du repository

```

try (EntityManager em =
emf.createEntityManager()) {
    CustomerRepository
customerRepository = new
CustomerRepository(em);
}

```

```

List<Customer> customers
= customerRepository.findByStatus(1,
5, Customer.Status.LEAD);
customers.forEach(c ->
System.out.println(c.getFirstname()
+ " " + c.getLastname()));
}

```

Cody Lowe
Austin Mante
Ta Corwin
Vanita Hodkiewicz
Geri Kuvalis

10 Interrogation avec JPA : Points clés

- Approches disponibles
 - JPQL : langage orienté objet
 - Criteria API : requêtes typées
 - SQL natif : performances
 - Named Queries : réutilisation
- Bonnes pratiques
 - Pagination systématique
 - Entity Graphs pour les associations
 - DTOs pour les projections
 - Repositories pour l'encapsulation