

# Programació



UT14.3 Consultes

# Definició de consultes

L'API JPA proporciona la interfície `Query` per a configurar i executar consultes. Podem obtindre una instància que implemente eixa interfície mitjançant els mètodes de l'`EntityManager`: `createNativeQuery`, `createQuery` i `createNamedQuery`. També disposa de la interfície `TypedQuery<T>` que ens permet un major control de tipus.

- `createNativeQuery` crea una instància `Query` per a executar una instrucció SQL nativa. Si la consulta no és d'actualització o eliminació, l'execució de la consulta farà que cada fila del resultat SQL es retorne com un `Object[]` (o `Object` si només hi ha una columna). El valor de les columnes es retornen en l'ordre en què apareixen a la llista de selecció i s'apliquen els tipus de dades JDBC per defecte.

```
Query query = em.createNativeQuery("SELECT * FROM categories ORDER BY id");
```

# Definició de consultes

- *createQuery* crea una instància Query o TypedQuery per a executar una instrucció JPQL

```
Query query = em.createQuery("SELECT c FROM Categories c WHERE c.id=51");
TypedQuery<Categories> query2 = em.createQuery("SELECT c FROM Categories c WHERE c.id=51", Categories.class);
```

- *createNamedQuery* crea una instància Query o TypedQuery per a executar una consulta “amb nom” que s’haurà definit a l’entitat.

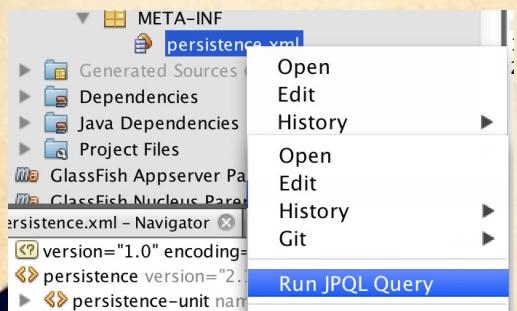
```
19  * @author ciclost
20  */
21  @Entity
22  @Table(name = "CATEGORIES")
23  @NamedQueries({
24      @NamedQuery(name = "Categories.findAll", query = "SELECT c FROM Categories c"),
25      @NamedQuery(name = "Categories.findById", query = "SELECT c FROM Categories c WHERE c.id = :id"),
26      @NamedQuery(name = "Categories.findByNom", query = "SELECT c FROM Categories c WHERE c.nom = :nom")})
27
28  public class Categories implements Serializable {
29      @Id
```

```
Query query = em.createNamedQuery("Categories.findById");
TypedQuery<Categories> query2 = em.createNamedQuery("Categories.findById", Categories.class);
```

# JPQL

Jakarta (anteriorment Java) Persistence Query Language és el llenguatge en el qual es construeixen les consultes en JPA. Encara que en aparença és molt similar a SQL, en la realitat operen en mons totalment diferents. En **JPQL** es **construeixen sobre classes** i entitats, mentre que SQL opera sobre taules, columnes i files en la base de dades.

Encara que SQL està estandarditzat, cada implementació (Oracle, MySQL, PostgreSQL, SQLServer...) té les seues xicotetes particularitats. Això no ocorre amb JPQL, que és totalment independent del SGBD, de la plataforma i del proveïdor que l'implementa.



**JPQL Query0**

Persistence Unit: AntJPAPU

```
SELECT c FROM Categories c
```

Result SQL

```
SELECT ID, NOM FROM CATEGORIES
```

**JPQL Query0**

Persistence Unit: AntJPAPU

```
SELECT c FROM Categories c
```

Result SQL

id	nom
1	Teclats
51	Impresores
101	Monitors
151	Emmagatzematge
201	Consoles

**JPQL Query0**

Persistence Unit: AntJPAPU

```
SELECT c.nom  
FROM Categories c  
WHERE c.nom LIKE "%es%"
```

Result SQL

Column
Impresores
Consoles

# JPQL. SELECT

La consulta més senzilla en JPQL és la que selecciona totes les instàncies d'un únic tipus d'entitat. Per exemple podem consultar tots els empleats de la BD:

```
SELECT e FROM Empleat e
```

En JPQL **se seleccionen instàncies d'un tipus d'entitat** del model del domini. La clàusula SELECT és lleugerament diferent a la d'SQL llistant només l'àlies "e" de l'Empleat. Este tipus indica el tipus de dades que retornarà la consulta, una llista de zero o més instàncies Empleat.

# JPQL. SELECT

A partir de l'àlies, podem obtindre els seus atributs i recórrer les relacions en les quals participa utilitzant l'operador punt (.). Per exemple, si només volem els noms dels empleats podem definir la següent consulta:

```
SELECT e.nom FROM Empleat e
```

Ja que l'entitat Empleat té un camp persistent anomenat "nom" de tipus String, esta consulta retornarà una llista de zero o més objectes de tipus String. Podem també seleccionar una entitat el tipus de la qual no llistem en la clàusula SELECT, utilitzant els atributs relació. Per exemple:

```
SELECT e.departament FROM Empleat e
```

A causa de la relació molts-a-un entre Empleat i Departament la consulta retornarà una llista d'instàncies de tipus Departament.

# JPQL. SELECT

També disposem de clàusules opcionals com WHERE, ORDER BY, GROUP BY o HAVING. Vegem com obtindre tots els empleats el sou dels quals és superior a 1200 euros, ordenats pel nom:

```
SELECT e  
FROM Empleat e  
WHERE e.sou >= 1200  
ORDER BY e.nom
```

Analitza el següent codi JPQL:

```
SELECT e  
FROM Empleat e  
WHERE e.departament.nom = 'INF'  
AND e.direccio.provincia IN ('ALC', 'VAL')
```

# JPQL. Execució d'una consulta

Ja hem vist que Query gestiona una consulta. Podem executar-la:

```
public class AntJPA {  
  
    /**  
     * @param args the command line arguments  
     */  
    public static void main(String[] args) {  
  
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("AntJPAPU");  
        EntityManager em = emf.createEntityManager();  
  
        Query query = em.createQuery("SELECT c FROM Categories c WHERE c.id = '101'");  
  
        System.out.println(query.getSingleResult());  
        System.out.println(query.getResultList());  
    }  
}
```

tjpa.AntJPA >

```
- AntJPA (run) ×  
  
run:  
antjpa.Categories[ id=101 nom=Monitors]  
[antjpa.Categories[ id=101 nom=Monitors]]  
BUILD SUCCESSFUL (total time: 10 seconds)
```

```
    */  
    public static void main(String[] args) {  
  
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("AntJPAPU");  
        EntityManager em = emf.createEntityManager();  
  
        Query query = em.createQuery("SELECT c FROM Categories c WHERE c.nom LIKE '%ses%'");  
  
        List<Categories> llista = query.getResultList();  
        for(Categories categoria: llista){  
            System.out.println("Categoria: " + categoria);  
        }  
    }  
}
```

antjpa.AntJPA >

out - AntJPA (run) ×

```
run:  
Categoria: antjpa.Categories[ id=51 nom=Impresores]  
Categoria: antjpa.Categories[ id=201 nom=Consoles]  
BUILD SUCCESSFUL (total time: 10 seconds)
```

# JPQL. Execució d'una consulta

És possible limitar els resultats, definint una quantitat màxima d'instàncies a retornar en la consulta amb *setMaxResult* i un número d'instància a partir del qual es construeix la llista amb *setFirstResult*:

```
public static void main(String[] args) {  
  
    EntityManagerFactory emf = Persistence.createEntityManagerFactory("AntJPAPU");  
    EntityManager em = emf.createEntityManager();  
  
    Query query = em.createQuery("SELECT c FROM Categories c ORDER BY c.nom");  
  
    for(Object categoria: query.getResultList()){  
        System.out.println("Categoria: " + categoria);  
    }  
  
}
```

```
antjpa.AntJPA >  
out - AntJPA (run) x  
run:  
Categoria: antjpa.Categories[ id=201 nom=Consoles]  
Categoria: antjpa.Categories[ id=151 nom=Emmagatzematge]  
Categoria: antjpa.Categories[ id=51 nom=Impresores]  
Categoria: antjpa.Categories[ id=101 nom=Monitors]  
Categoria: antjpa.Categories[ id=1 nom=Teclats]  
BUILD SUCCESSFUL (total time: 12 seconds)
```

```
/**  
 * @param args the command line arguments  
 */  
public static void main(String[] args) {  
  
    EntityManagerFactory emf = Persistence.createEntityManagerFactory("AntJPAPU");  
    EntityManager em = emf.createEntityManager();  
  
    Query query = em.createQuery("SELECT c FROM Categories c ORDER BY c.nom");  
    query.setFirstResult(1);  
    query.setMaxResults(3);  
  
    for(Object categoria: query.getResultList()){  
        System.out.println("Categoria: " + categoria);  
    }  
  
}
```

```
antjpa.AntJPA >  
out - AntJPA (run) x  
run:  
Categoria: antjpa.Categories[ id=151 nom=Emmagatzematge]  
Categoria: antjpa.Categories[ id=51 nom=Impresores]  
Categoria: antjpa.Categories[ id=101 nom=Monitors]  
BUILD SUCCESSFUL (total time: 12 seconds)
```

# JPQL. Altres consultes amb SELECT

SELECT permet seleccionar atributs o utilitzar funcions agregades. Vegem com seleccionar només els noms dels empleats:

```
SELECT e.nom FROM Empleat e
```

En este exemple, l'execució de la consulta retornarà una llista d'Object, ja que l'atribut nom, encara que siga del tipus que siga, es retorna com a Object:

```
List<Object> totsElsNoms = query.getResultList();
for(Object nom: totsElsNoms){
    System.out.println(nom);
}
```

# JPQL. Altres consultes amb SELECT

Si ara volem un informe amb el nom i el sou de cada empleat:

```
SELECT e.nom, e.sou FROM Empleat e
```

El mètode getResultList() retornarà una llista d'arrays de tipus Object, es a dir, una llista d'Object[] on cada atribut tornat ocupa l'index corresponent a la seua posició en la consulta:

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

Query query = em.createQuery("SELECT e.nom, e.sou FROM Empleat e");
List<Object[]> informe = query.getResultList();
for(Object registre[]: informe){
    System.out.print("Nom: " + aux[0]);
    System.out.print(" - Sou: " + aux[1]);
    System.out.println();
}
```

# JPQL. Consultes parametritzades

Fins ara hem utilitzat consultes estàtiques. En ocasions és interessant parametritzar una consulta per a obtindre resultats distints segons canviem els paràmetres. JPQL disposa de **consultes parametritzades** també anomenades **consultes dinàmiques**, que inclouen certs paràmetres als quals assignarem valors, adaptant la mateixa consulta a les necessitats de cada moment. JPQL suporta **dos tipus de sintaxi per a la lligadura** (binding) de paràmetres: **posicional** (?) i **per nom** (:). Vegem un exemple de cada cas:

```
/*
public static void main(String[] args) {

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

    String jpql = "SELECT e FROM Empleat e"
        + "WHERE e.carrec = ?1 AND"
        + "e.sou >= ?2";

    Query query = em.createQuery(jpql);
    /* Assignem valors als paràmetres per a consultar tots els gerents
       que guanyen més de 2000 euros */
    query.setParameter(1, "Gerent");
    query.setParameter(2, 2000);

    // Ara podem executar la consulta i guardar-la en una llista
    // List<Empleat> resultat = query.getResultList();
}

}
```

```
/*
public static void main(String[] args) {

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

    String jpql = "SELECT e FROM Empleat e"
        + "WHERE e.carrec = :carrec AND"
        + "e.sou >= :quantitat";

    Query query = em.createQuery(jpql);
    /* Assignem valors als paràmetres per a consultar tots els gerents
       que guanyen més de 2000 euros */
    query.setParameter("carrec", "Gerent");
    query.setParameter("quantitat", 2000);

    // Ara podem executar la consulta i guardar-la en una llista
    // List<Empleat> resultat = query.getResultList();
}

}
```

# JPQL. Consultes parametritzades

Analitza el següent codi JPQL:

```
SELECT p.empleat, COUNT(p)
FROM Projecte p
WHERE p.dataCreacio is BETWEEN :data1 and :data2
GROUP BY p.empleat
HAVING COUNT(p) > 5
```

# JPQL. Subconsultes, nuls i buits.

Podem niar múltiples consultes, per exemple:

```
SELECT e FROM Empleat e WHERE e.proyecto IN  
    (SELECT p FROM proyecto p WHERE p.tipus = 'A')
```

Per altra banda, es poden utilitzar els operadors IS NULL o IS NOT NULL per a comprovar si un atribut és o no nul:

```
WHERE e.departament IS NOT NULL
```

En el cas de les col·leccions (per exemple una llista) cal utilitzar IS EMPTY o IS NOT EMPTY

```
WHERE e.projects IS EMPTY
```

Esbrina que farà el següent codi JPQL

```
SELECT d FROM Departament d  
WHERE :empleat NOT MEMBER OF d.empleats
```

Esbrina que farà el següent codi JPQL

```
SELECT d FROM Departament d  
WHERE SIZE(d.empleats) = 1
```

# JPQL. Joins entre entitats

Igual que en SQL és possible definir consultes que realitzen una selecció en el resultat d'unir (join) entitats entre les quals s'ha establit una relació. Per exemple, el següent codi mostra un join entre les entitats Empleat i CompteCorreu per a recuperar tots els correus electrònics d'un departament específic:

```
SELECT c.correu FROM Empleat e, CompteCorreu c  
WHERE e = c.empleat AND e.departament.nom = 'INF'
```

També és possible utilitzar l'operador JOIN. Un avantatge és que el join pot especificar-se en termes de la pròpia associació i que el motor de consultes proporcionarà automàticament el criteri de join necessari quan genere el SQL:

```
SELECT c.correu FROM Empleat e JOIN e.comptesCorreu c  
WHERE e.departament.nom = 'INF'
```

(\*) JPQL suporta múltiples tipus de joins, incloent inner i outer joins, left joins i una tècnica denominada fetch joins per a carregar dades associades a les entitats que no es retornen directament. En este curs no aprofundirem més sobre això.

# JPQL. LIKE

L'operador LIKE permet buscar un patró en un text i comprova si una cadena especificada compleix un patró específicat. Si es precedeix amb l'operador NOT retorna aquells valors que no compleixen el patró. El patró pot incloure qualsevol caràcter i els següents caràcters lliures:

- El caràcter **tant per cent (%)** aparella amb 0 o més caràcters qualssevol.
- El **caràcter subratllat (\_)** aparella un únic caràcter qualssevol.

L'operand esquerre és sempre la cadena que es comprova i el dret el patró. Exemples:

- `pais.nom LIKE '_r%'` és TRUE per a 'Brasil' i és FALSE per a 'Dinamarca'
- `pais.nom LIKE '%'` és sempre TRUE.
- `pais.nom NOT LIKE '%'` és sempre FALSE.

# JPQL. Consultes amb nom

Com hem vist abans les consultes amb nom es defineixen juntament amb l'entitat, utilitzant l'anotació **@NamedQuery**. Milloren el rendiment ja que són processades una única volta.

L'anotació **@NamedQuery** defineix mitjançant els paràmetres “**name**” i “**query**” una consulta amb nom en una entitat. Observem uns exemples d’ús:

Implementació dels NamedQuery:

```
@Entity
@NamedQueries({
    @NamedQuery(name="Empleat.tots", query="SELECT e FROM Empleat e"),
    @NamedQuery(name="Empleat.carrec",
                query="SELECT e FROM Empleat e WHERE e.carrec = :carrec"),
    @NamedQuery (name = "Empleat.carrecOficina",
                query= "SELECT e FROM Empleat e WHERE e.carrec = :carrec AND e.oficina = :oficina")
})
public class Empleat implements Serializable{
```

```
Query query1 = em.createNamedQuery("Empleats.tots");
List<Empleat> tots = query1.getResultList();
```

```
Query query2 = em.createNamedQuery("Empleat.carrec");
query2.setParameter("carrec", "Gerent");
List<Empleat> gerents = query2.getResultList();
```

```
Query query3 = em.createNamedQuery("Empleat.carrecOficina");
query3.setParameter("carrec", "Informàtic");
query3.setParameter("oficina", 12);
List<Empleat> informaticsOficina12 = query3.getResultList();
```

Ús de les consultes NamedQuery:

# JPQL. UPDATE i DELETE

JPQL permet realitzar sentències d'actualització o esborrat d'objectes. A continuació es mostra com augmentar un 10% el sou de tots els empleats el nom dels quals comence per "J".

```
UPDATE Empleat e SET e.sou = 1.1 * e.sou WHERE e.nom LIKE 'F%'
```

La sintaxi de DELETE és molt similar a la d'SQL. Vegem com eliminar tots els empleats que ocupen un càrrec de gerent.

```
DELETE FROM Empleat e WHERE e.carrec = 'Gerent'
```

També podem utilitzar consultes parametritzades o dinàmiques:

```
DELETE FROM Empleat e WHERE e.carrec = :carrec
```

# JPQL. UPDATE i DELETE

L'execució de sentències UPDATE i DELETE difereixen de l'execució de les SELECT:

- No retornen cap resultat, per tant en comptes del mètode "getResultSet" usarem "executeUpdate".
- "executeUpdate" retorna la quantitat de files afectades per la consulta.
- Impliquen un canvi en la BD, per la qual cosa necessiten executar-se dins d'una transacció

Per exemple per a eliminar els gerents, el codi Java quedaria:

```
Query query = em.createQuery("DELETE FROM Empleat e WHERE e.carrec = 'Gerent'");  
EntityTransaction tx = em.getTransaction();  
tx.begin();  
int quantsBorrats = query.executeUpdate();  
tx.commit();  
System.out.println("Hem eliminat " + quantsBorrats + " gerents.");
```