



Frameworks de persistencia - JPA

Sesión 5 - Consultas con JPQL



Índice

- Introducción
- Consultas dinámicas y named
- Ejemplos de entidades
- SELECT
- WHERE
- JOINS
- GROUP BY
- Consultas en los DAO



JPQL

- Tiene una sintaxis muy similar a SQL
- La cláusula básica es SELECT, que puede devolver una lista de valores o un único valor
- Los valores devueltos pueden ser datos básicos (tipos de los atributos) o entidades (resultantes de relaciones definidas entre las entidades)
- Podemos filtrar los datos devueltos para que complan las condiciones definidas con la cláusula WHERE
- Es posible proyectar el SELECT y devolver tuplas como resultados
- Soporta JOINS entre las entidades para formular las condiciones del SELECT



Definiendo y ejecutando las consultas

- Dos formas de definir consultas:
 - Dinámicas con el método `createQuery()` del entity manager
 - Estáticas, definiéndolas en la entidad y asociándoles un nombre
- Para ejecutar un consulta hay que llamar `setParameter()` para definir los parámetros y a `getSingleResult()` o `getResultList()` dependiendo de si se devuelve un único valor o una lista
- Las queries devuelven colecciones de entidades gestionadas
- Hay que hacer casting de los objetos resultantes



Ejemplos consultas dinámicas

```
public double salarioPorDeptoNombre(String nombreDepto, String nombreEmpleado) {
    final String SALARIO_POR_DEPTO_NOMBRE = "SELECT e.sueldo " +
        "FROM Empleado e " +
        "WHERE e.departamento.nombre = :deptNombre AND " +
        "       e.nombre = :empNombre";

    EntityManager em = emf.createEntityManager();
    em.getTransaction().begin();

    TypedQuery<Double> query = em.createQuery(
        SALARIO_POR_DEPTO_NOMBRE, Double.class)
        .setParameter("deptNombre", nombreDepto)
        .setParameter("empNombre", nombreEmpleado);
    double sueldo = query.getSingleResult();
    System.out.println(sueldo);
    em.getTransaction().commit();
    em.close();
    return sueldo;
}
```



Ejemplo consultas con nombre

```
@Entity
@NamedQueries({
    @NamedQuery(name="Empleado.findAll",
        query="SELECT e FROM Empleado e"),
    @NamedQuery(name="Empleado.findById",
        query="SELECT e FROM Empleado e WHERE e.id = :id"),
    @NamedQuery(name="Empleado.findByName",
        query="SELECT e FROM Empleado e WHERE e.nombre = :nombre")
})
public class Empleado {
    ...
}
```

```
public Empleado findEmpleadoPorNombre(String nombreEmpleado) {
    EntityManager em = emf.createEntityManager();
    em.getTransaction().begin();
    TypedQuery<Empleado> queryEmpleado =
        em.createNamedQuery("Empleado.findByName", Empleado.class)
            .setParameter("nombre", nombreEmpleado);
    Empleado empleado = queryEmpleado.getSingleResult();
    em.getTransaction().commit();
    em.close();
    return empleado;
}
```



Parámetros en las consultas

- Posicional

```
QUERY = "SELECT e  
FROM Empleado e  
WHERE e.departamento = ?1 AND  
e.salario > ?2"
```

```
List<Empleado> result = em.createQuery(  
    "SELECT e " +  
    "FROM Proyecto p JOIN p.empleados e " +  
    "WHERE p.nombre = ?1")  
    .setParameter(1, nombreProyecto)  
    .getResultList();
```

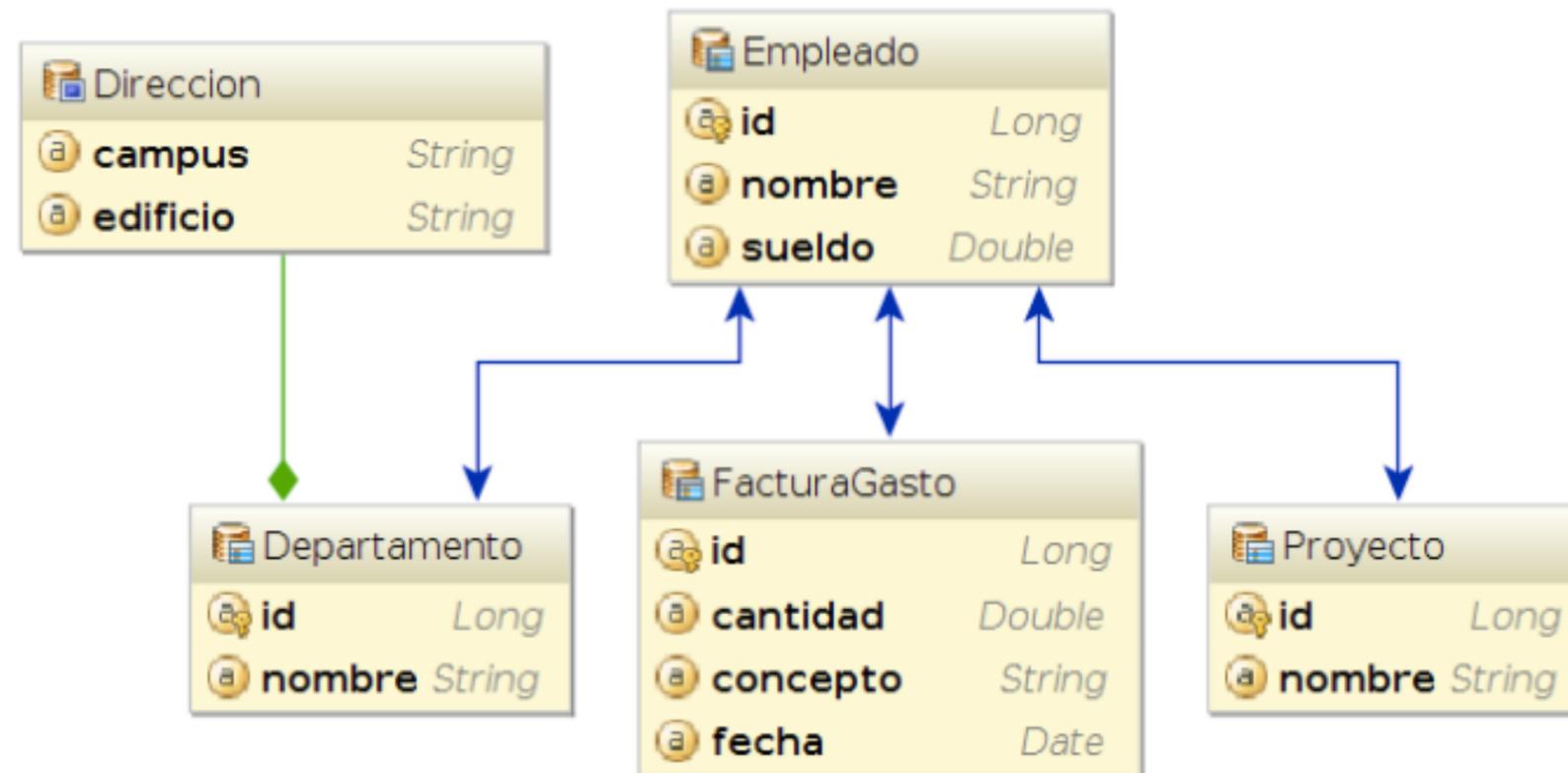
- Por nombre

```
QUERY = "SELECT e  
FROM Empleado e  
WHERE e.departamento = :dept AND  
e.salario > :sal"
```

```
Empleado empleado =  
    em.createNamedQuery("Empleado.findByNombre")  
        .setParameter("nombre", nombre)  
        .getSingleResult();
```



Entidades que vamos a usar en las consultas





Definiciones en JPA

```
@Entity
public class Empleado {
    @Id
    private Long id;
    private String nombre;
    private Double sueldo;
    @ManyToOne
    private Departamento departamento;
    @ManyToMany
    private Set<Proyecto> proyectos;
    @OneToMany(mappedBy = "empleado")
    private Set<FacturaGasto> gastos;
    ...
}
```

```
@Entity
public class Departamento {
    @Id
    private Long id;
    private String nombre;
    @Embedded
    private Direccion direccion;
    @OneToMany(mappedBy = "departamento")
    private Set<Empleado> empleados;
    ...
}
```

```
@Entity
public class Proyecto {
    @Id
    private Long id;
    private String nombre;
    @ManyToMany(mappedBy = "proyectos")
    private Set<Empleado> empleados;
    ...
}
```

```
@Entity
public class FacturaGasto {
    @Id
    private Long id;
    @ManyToOne
    private Empleado empleado;
    private String concepto;
    private Double cantidad;
    private Date fecha;
    ...
}
```

```
@Embeddable
public class Direccion {
    private String campus;
    private String edificio;
    ...
}
```



SELECT (1)

- Seleccionamos todas las instancias de una entidad

```
SELECT e FROM Empleado e
```

```
final String FIND_ALL_EMPLEADOS = "SELECT e FROM Empleado e ";
TypedQuery<Empleado> query = em.createQuery(FIND_ALL_EMPLEADOS, Empleado.class);
List<Empleado> empleados = query.getResultList();
if (empleados != null) {
    for (Empleado empleado : empleados) {
        System.out.println(empleado.getNombre() + " - " +
            empleado.getDespacho().getCodigoDespacho());
    }
}
```

- Seleccionamos todos los posibles valores de un atributo de la entidad

```
SELECT e.sueldo FROM Empleado e
```

```
final String FIND_ALL_SUELDOS_EMPLEADOS = "SELECT e.sueldo FROM Empleado e ";
TypedQuery<Double> query = em.createQuery(FIND_ALL_SUELDOS_EMPLEADOS, Double.class);
List<Double> sueldosEmpleados = query.getResultList();
if (sueldosEmpleados != null) {
    for (double sueldoEmpleado : sueldosEmpleados) {
        System.out.println(sueldoEmpleado);
    }
}
```



SELECT (2)

- Seleccionamos todas las entidades (con repetición) asociadas con una instancia

```
SELECT e.departamento FROM Empleado e
```

- Algunos modificadores: SELECT DISTINCT and ORDER BY

```
SELECT DISTINCT e.departamento FROM Empleado e ORDER BY e.nombre
```

- Paginación de resultados

```
Query q = em.createQuery("SELECT e FROM Empleado e");  
q.setFirstResult(20);  
q.setMaxResults(10);  
List empleados = q.getResultList();
```



WHERE

- La mayoría de operadores de SQL están en JPQL, incluyendo los operadores `IN`, `LIKE` y `BETWEEN`, llamadas a funciones como `SUBSTRING` o `LENGTH` y subqueries

```
SELECT e
  FROM Empleado e
   WHERE e.departamento.nombre LIKE '%IA' AND
         e.sueldo BETWEEN 2000 AND 2500
```

```
WHERE e.departamento IS NOT NULL
```

```
WHERE e.proyectos IS EMPTY
```



Pattern matching

- JPAQL contiene un conjunto de funciones muy útiles para realizar pattern matching con expresiones textuales
- LIKE permite buscar patrones
 - El carácter '%' empareja con 0 o más caracteres
 - El carácter '_' empareja con sólo 1 carácter

```
LIKE '_r%' es TRUE para 'Brasil' y false para 'Dinamarca'
```

- CONCAT concatena dos cadenas

```
WHERE CONCAT(e.nombre, e.apellido) LIKE 'Ju%cia'
```

- LOWER y UPPER convierten una cadena en mayúsculas y minúsculas

```
WHERE UPPER(CONCAT(l.autor, l.titulo)) LIKE UPPER(:patron)
```

- Otras funciones: LOCATE, TRIM, SUBSTRING, LENGTH



Proyecciones

- El resultado de una consulta pueden ser tuplas
- Cada n-tupla se implementa como un array de n componentes

```
SELECT e.nombre, e.salario
FROM Empleado e
```

```
result[0] - nombre
result[1] - salario
```

```
final String FIND_NOMBRE_SUELDO =
    "SELECT e.nombre, e.sueldo FROM Empleado e";

TypedQuery<Object[]> query = em.createQuery(
    FIND_NOMBRE_SUELDO, Object[].class);

List<Object[]> tuplas = query.getResultList();

if (tuplas != null) {
    for (Object[] tupla : tuplas) {
        String nombre = (String) tupla[0];
        Double sueldo = (Double) tupla[1];
        System.out.println(nombre + " - " + sueldo);
    }
}
```



Joins entre entidades

- Es posible definir las condiciones sobre el resultado de unir entidades entre las que hay una relación
- La consulta se mapeará en una consulta similar en SQL
- Devolvemos todos los conceptos de gastos de empleados del 'DCCIA'

```
SELECT f.concepto
FROM Empleado e, FacturaGasto f
WHERE e = f.empleado AND
      e.departamento.nombre='DCCIA'
```

```
SELECT f.concepto
FROM Empleado e JOIN e.gastos f
WHERE e.departamento.nombre='DCCIA'
```



Más ejemplos de joins (1)

- Selecciona todos los departamentos distintos asociados a empleados

```
SELECT DISTINCT e.departamento FROM Empleado e
```

```
SELECT DISTINCT d FROM Empleado e JOIN e.departamento d
```

- Selecciona los proyectos distintos que pertenecen a empleados del departamento DLSI

```
SELECT DISTINCT p  
  FROM Departamento d JOIN d.empleados e JOIN e.proyectos p  
 WHERE d.nombre= 'DLSI'
```



Más ejemplos de joins (2)

- Selecciona los departamentos en el campus UA en donde trabajan empleados que participan en el proyecto Reconocimiento de caras:

```
SELECT DISTINCT d
  FROM Empleado e JOIN e.departamento d JOIN e.proyectos p
 WHERE d.direccion.campus='UA' AND
        p.nombre='Reconocimiento de caras'
```



Subqueries

- Es posible anidar múltiples queries
- Ejemplo: suponemos una relación bidireccional uno-a-muchos entre Empleados y Proyectos
- Para obtener los empleados que participan en proyectos de tipo 'A':

```
SELECT e FROM Empleado e
WHERE e.proyecto IN (SELECT p
                     FROM Proyecto p
                     WHERE p.tipo = 'A')
```



Agrupaciones

- Las cláusulas AVG, COUNT, MAX, MIN, SUM pueden aplicarse a grupos de valores; devuelven Long y Double
- Se pueden obtener los grupos utilizando GROUP BY y filtrar los resultados con HAVING

```
SELECT p.Empleado, COUNT(p)
FROM Proyecto p
GROUP BY p.Empleado
```

```
List result = em.createQuery("SELECT p.Empleado, COUNT(p)" +
"FROM Proyecto p GROUP BY p.Empleado").getResultList();
Iterator res = result.iterator();
while (res.hasNext()) {
    Object[] tupla = (Object[]) res.next();
    Empleado emp = (Empleado) tupla[0];
    long count = ((Long) tupla[1]).longValue();
    ...
}
```



Un último ejemplo

- Devuelve todos los empleados que participan en más de 5 proyectos creados entre dos fechas

```
SELECT p.Empleado, COUNT(p)
FROM Proyecto p
WHERE p.fechaCreacion is BETWEEN :date1 and :date2
GROUP BY p.Empleado
HAVING COUNT(p) > 5
```



API Criteria

- Introducidas en JPA 2.0
- Las queries se construyen paso a paso y el compilador puede asegurar que son correctas sintácticamente
- El API es bastante avanzado y necesitaríamos una sesión completa para introducir sus elementos básicos, sólo vamos a ver un ejemplo sencillo, para comprobar su estilo

```
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Empleado> cq =
    cb.createQuery(Empleado.class);
Root<Empleado> e = cq.from(Empleado.class);
cq.select(e);
cq.where(cb.greaterThan(e.get("sueldo"), 2500.0));
Query query = em.createQuery(cq);
List<Empleado> emps = query.getResultList();
```



Organización de las queries en los DAOs

- El DAO que gestiona una entidad es un buen lugar donde incluir sus queries. Por ejemplo: EmpleadoDAO:

```
public class EmpleadoDao extends Dao<Empleado, Long> {
    String FIND_ALL_EMPLEADOS = "SELECT e FROM Empleado e ";

    String FIND_ALL_SUELDOS_EMPLEADOS =
        "SELECT e.sueldo FROM Empleado e";

    String FIND_PROYECTOS_EMPLEADOS_DEPARTAMENTO =
        "SELECT DISTINCT p" +
        " FROM Departamento d JOIN d.empleados e JOIN
e.proyectos p" +
        " WHERE d.nombre='DLSI'";

    public EmpleadoDao(EntityManager em) {
        super(em);
    }

    @Override
    public Empleado find(Long id) {
        EntityManager em = this.getEntityManager();
        return em.find(Empleado.class, id);
    }
}
```

```
public List<Empleado> listAllEmpleados() {
    EntityManager em = this.getEntityManager();
    TypedQuery<Empleado> query = em.createQuery(
        FIND_ALL_EMPLEADOS, Empleado.class);
    return query.getResultList();
}

public List<Double> listAllSueldosEmpleados() {
    EntityManager em = this.getEntityManager();
    TypedQuery<Double> query = em.createQuery(
        FIND_ALL_SUELDOS_EMPLEADOS, Double.class);
    return query.getResultList();
}

public List<Proyecto> listAllProyectosEmpleadosDepartamento(String
nombreDepto) {
    EntityManager em = this.getEntityManager();
    TypedQuery<Proyecto> query = em.createQuery(
        FIND_PROYECTOS_EMPLEADOS_DEPARTAMENTO, Proyecto.class);
    return query.getResultList();
}
}
```



¿Preguntas?