Consultas. Arquitectura de una aplicación JPA

Índice

1 Consultas en JPA....2
1.1 Java Persistence Query Language....2
1.2 Definición de consultas....4
1.3 Ejecución de consultas....6

1. Consultas en JPA

1.1. Java Persistence Query Language

1.1.1. Comenzando

La consulta más sencilla en JPQL es la que selecciona todas las instancias de un único tipo de entidad:

```
SELECT e
FROM Empleado e
```

La sintaxis de JPQL es similar a la de SQL. De esta forma los desarrolladores con experiencia en SQL pueden comenzar a utilizarlo rápidamente. La diferencia principal es que en SQL se seleccionan filas de una tabla mientras que en JPQL se seleccionan instancias de un tipo de entidad del modelo del dominio. La cláusula `SELECT` es ligeramente distinta a la de SQL listando sólo el alias `e` del `Empleado`. Este tipo indica el tipo de datos que va a devolver la consulta, una lista de cero o más instancias `Empleado`.

A partir del alias, podemos obtener sus atributos y recorrer las relaciones en las que participa utilizando el operador punto (.). Por ejemplo, si sólo queremos los nombres de los empleados podemos definir la siguiente consulta:

```
SELECT e.nombre
FROM Empleado e
```

Ya que la entidad `Empleado` tiene un campo persistente llamado `nombre` de tipo `String`, esta consulta devolverá una lista de cero o más objetos de tipo `String`.

Podemos también seleccionar una entidad cuyo tipo no listamos en la cláusula `SELECT`, utilizando los atributos relación. Por ejemplo:

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

Debido a la relación muchos-a-uno entre `Empleado` y `Departamento` la consulta devolverá una lista de instancias de tipo `Departamento`.

1.1.2. Filtrando los resultados

Igual que SQL, JPQL contiene una cláusula `WHERE` para especificar condiciones que deben cumplir los datos que se devuelven. La mayoría de los operadores disponibles en SQL están en JPQL, incluyendo las operaciones básicas de comparación: `IN`, `LIKE` y `BETWEEN`, funciones como `SUBSTRING` o `LENGTH` y subqueries. Igual que antes, la diferencia fundamental es que se utilizan expresiones sobre entidades y no sobre columnas. El siguiente código muestra un ejemplo:

```
SELECT e
FROM Empleado e
WHERE e.departamento.name = 'NA42' AND
      e.direccion.provincia IN ('ALC','VAL')
```

1.1.3. Proyectando los resultados

Es posible recuperar sólo alguno de los atributos de las instancias. Esto es útil cuando tenemos una gran cantidad de atributos (columnas) y sólo vamos a necesitar listar algunos. Por ejemplo, la siguiente consulta selecciona sólo el nombre y el salario de los empleados:

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

El resultado será una colección de cero o más instancias de arrays de tipo `Object`. Cada array contiene dos elementos, el primero un `String` y el segundo un `Double`.

1.1.4. Joins entre entidades

Al igual que en SQL es posible definir consultas que realicen una selección en el resultado de unir (*join*) entidades entre las que se ha establecido una relación. Por ejemplo, el siguiente código muestra un join entre las entidades `Empleado` y `CuentaCorreo` para recuperar todos los correos electrónicos de un departamento específico:

```
SELECT c.correo
FROM Empleado e, CuentaCorreo c
WHERE e = c.empleado AND
      e.departamento.nombre = 'NA42'
```

En JPQL también es posible especificar joins en la cláusula `FROM` utilizando el operador `JOIN`. Una ventaja de este operador es que el join puede especificarse en términos de la propia asociación y que el motor de consultas proporcionará automáticamente el criterio de join necesario cuando genere el SQL. El siguiente código muestra la misma consulta reescrita para utilizar este operador:

```
SELECT c.correo
FROM Empleado e JOIN e.cuentasCorreo c
WHERE e.departamento.nombre = 'NA42'
```

JPQL soporta múltiples tipos de joins, incluyendo inner y outer joins, left joins y una técnica denominada fetch joins para cargar datos asociados a las entidades que no se devuelven directamente. No tenemos tiempo de detallar todos ellos. Vamos a ver algunos ejemplos más, para tener una idea de la potencia de la sintaxis.

Selecciona todos los departamentos distintos asociados a empleados:

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

Otra forma de hacer la misma consulta utilizando el operador `JOIN`:

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

Selecciona los departamentos distintos que trabajan en Alicante y que participan en el proyecto 'BlueBook':

```
SELECT DISTINCT e.departamento
FROM Proyecto p JOIN p.empleados e
WHERE p.nombre = 'BlueBook' AND
      e.direccion.localidad = 'ALC'
```

Selecciona los proyectos distintos que pertenecen a empleados de un departamento:

```
SELECT DISTINCT p
FROM Departamento d JOIN d.empleados JOIN e.proyectos p
```

Selecciona todos los empleados y recupera la entidad `Direccion` con la que está relacionado cada uno:

```
SELECT e
FROM Empleado e JOIN FETCH e.direccion
```

1.1.5. Parámetros de las consultas

JPQL soporta dos tipos de sintaxis para la ligadura (*binding*) de parámetros: posicional y por nombre. Vemos un ejemplo del primer caso:

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

Veremos en el siguiente apartado como ligar parámetros determinados a esas posiciones. La segunda forma de definir parámetros es por nombre:

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

1.2. Definición de consultas

El API JPA proporciona la interfaz `Query` para configurar y ejecutar consultas. Podemos obtener una instancia que implemente esa interfaz mediante un dos de métodos del entity manager: `createQuery()` y `createNamedQuery()`. El primer método se utilizar para crear una consulta dinámica y el segundo una consulta con nombre.

Una vez obtenida la consulta podemos pasarle los parámetros con `setParameter()` y ejecutarla. Se definen dos métodos para ejecutar consultas: el método

`getSingleResult()` que devuelve un `Object` que es la única instancia resultante de la consulta y el método `getResultList()` que devuelve una lista de instancias resultantes de la consulta.

1.2.1. Consultas dinámicas

Se puede crear una consulta dinámica pasándole la cadena con la consulta al método `createQuery`. El proveedor de persistencia transforma en ese momento la consulta en el código SQL que se ejecutará en la base de datos. Veamos un ejemplo de consulta, que utiliza el paso de parámetros por nombre visto anteriormente:

```
public class EmpleadoDAO {
   private static final String QUERY =
      "SELECT e.salary " +
      "FROM Empleado e " +
      "WHERE e.departamento.nombre = : deptNombre AND " +
      "      e.nombre = : empNombre";

   // ...
   public long queryEmpleadoSalario(String empNombre, String
deptNombre) {
      return (Long) em.createQuery(QUERY)
                      .setParameter("deptNombre", deptNombre)
                      .setParameter("empNombre", empNombre)
                      .getSingleResult();
   }
}
```

Un inconveniente de las consultas dinámicas es que se procesan en tiempo de ejecución de la aplicación, con la consiguiente pérdida de rendimiento. Su ventaja principal es que se pueden definir con mucha comodidad.

1.2.2. Consultas con nombre

Las consultas con nombre se definen junto con la entidad, utilizando la anotación `@NamedQuery`. Mejoran el rendimiento con respecto a las consultas dinámicas, ya que son procesadas una única vez. El siguiente código muestra un ejemplo:

```
@NamedQuery(name="salarioPorNombreDepartamento",
         query="SELECT e.salary " +
               "FROM Empleado e " +
               "WHERE e.departamento.nombre = : deptNombre AND " +
               "      e.nombre = : empNombre")
```

Si se necesita más de una consulta para una entidad, deben incluirse en la anotación `@NamedQueries`, que acepta una array de una o más anotaciones `@NamedQuery`:

```
@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.findByNombre",
               query="SELECT e FROM Empleado e WHERE e.nombre =
:nombre")
})
```

Para ejecutar la consulta hay que llamar al método `createNamedQuery` pasándole como parámetro el nombre de la consulta. El siguiente código muestra un ejemplo:

```
public class EmpleadoDAO {
   // ...
   public Empleado findEmpleadoByNombre(String nombre) {
      return (Empleado)
em.createNamedQuery("Empleado.findByNombre")
                          .setParameter("nombre", nombre)
                          .getSingleResult();
   }

   public List<Empleado> findAll() {
      return (List<Empleado>)
em.createNamedQuery("Empleado.findAll")
                               .getResultList();
   }
}
```

1.3. Ejecución de consultas

Veamos por último los métodos del interfaz `Query` para ejecutar las consultas definidas. Son los métodos `getSingleResult()` y `getResultList()`.

Ambos métodos se deben lanzar sobre una `Query` ya construida y en la que se han introducido los parámetros. El método `getSingleResult()` se utiliza con consultas que devuelven un único resultado. Devuelve un `Object` que contiene el resultado de la consulta. Después de llamarlo conviene hacer un casting al tipo (entidad o tipo básico) que esperamos. Puede suceder que la consulta ejecutada no devuelva ningún resultado o devuelva más de uno. En el primer caso se genera la excepción `NoResultException` y en el segundo `NonUniqueResultException`.

Si no tenemos seguridad de una consulta vaya a devolver un único valor deberíamos llamar a `getResultList()`. Este método devuelve una `List` de `Object`. La utilización de la interfaz `List` en lugar de `Collection` es para soportar la devolución de colecciones ordenadas, generadas por consultas con la cláusula `ORDER BY`. En el caso en que la consulta no obtenga resultados, se devuelve una lista vacía. El siguiente código muestra un ejemplo de utilización de una consulta que devuelve una lista ordenada:

```
public void muestraEmpleadosProyecto(String nombreProyecto) {
   List<Empleado> result = em.createQuery(
                              "SELECT e " +
                              "FROM Proyecto p JOIN p.empleados e "
+
                              "WHERE p.nombre = ?1 " +
                              "ORDER BY e.name")
                              .setParameter(1, nombreProyecto)
```

```
                                .getResultList();
   for (empleado : result) {
      System.out.println(empleado.nombre);
   }
}
```