



UNIVERSIDAD DEL QUINDÍO
FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

Información general	
Actualización:	Carlos Andrés Flórez Villarraga
Duración estimada en minutos:	120
Docente:	Christian Andrés Candela y Einer Zapata
Guía no.	8
Nombre de la guía:	Java Persistence Api – Named Query

Información de la Guía

OBJETIVO

Aprender a usar las herramientas que proporciona JPA para la manipulación de las entidades que modelan nuestra aplicación.

CONCEPTOS BÁSICOS

Manejo de Eclipse, Java, Bases de Datos, DataSource, Entidades y GlassFish.

CONTEXTUALIZACIÓN TEÓRICA

JPA (Java Persistence Api), es el Api de Java que se encarga del manejo de persistencia de la aplicación. JPA es un framework ORM (Object Relational Mapping) el cual establece una relación entre los tipos de datos del lenguaje de programación y los usados por la base de datos. Adicionalmente, los ORM mapean las tablas a entidades (clases) en programación orientada a objetos logrando así, crear una base de datos orientada a objetos sobre la base de datos relacional. JPA proporciona un lenguaje (JPQL) para la realización de consultas sobre las entidades de forma independiente del motor de base de datos a ser usado para el almacenamiento de la información.

El lenguaje Java Persistence Query es una extensión del lenguaje de consulta de Enterprise JavaBeans (EJB QL). JPA adiciona las operaciones como eliminación, actualización, combinación (join), proyecciones, y subconsultas. Además, las consultas JPQL puede ser declarado de forma estática en los metadatos, o puede ser generado de forma dinámica en el código, para más información consultar el siguiente enlace: https://docs.oracle.com/cd/E11035_01/kodo41/full/html/ejb3_langref.html.

JPQL proporciona dos tipos de sentencia, la de consulta y la de modificación (actualización y borrado de datos). Las sentencias JPQL pueden ser:

SELECT

Es una cadena que está compuesta por las siguientes cláusulas.

- Una cláusula **select** que determina el tipo de objeto o valor a ser seleccionado.
- Una cláusula **from** que establece el dominio de la consulta (las entidades sobre las cuales se quiere realizar la consulta)
- Una cláusula **where** (opcional) que puede ser usada para filtrar (restringir) los resultados devueltos por la consulta.

- Una cláusula **group by** (opcional) que permite sumar los resultados de la consulta en términos de grupos.
- Una cláusula **havign** (opcional) que permite establecer filtros a ser aplicados sobre los grupos de resultados.
- Una cláusula **order by** (opcional) que permite ordenar los resultados de la consulta.

UPDATE | DELETE

Las sentencias de borrado o actualización de datos están compuestas por la sentencia (**update** o **delete**), y una cláusula **where**.

- Las cláusulas **update** o **delete** determinan el tipo de entidad a ser actualizada o borrada.
- La cláusula **where** puede ser usada para restringir el alcance de la cláusula de borrado o actualización.

NAMED QUERY

Los named query son anotaciones que permiten dar un nombre a una consulta que es usada de forma recurrente en el software, con la finalidad de poderla reutilizar a lo largo de la aplicación facilitando su modificación. Dicha anotación se crea en las entidades como por ejemplo:

```
@Entity
@NamedQueries({
    @NamedQuery(name = Persona.FIND_BY_ID, query = "select p from Persona p where p.cedula = :cedula"),
    @NamedQuery(name = Persona.GET_ALL, query = "select p from Persona p")
})
public class Persona implements Serializable {

    @Id
    private String cedula;

    /**
     * Variable que representa el atributo nombre de la Persona
     */
    @Column(length = 70)
    private String nombre;

    /**
     * Constante que identifica la consulta que permite buscar una
     * Persona por su cédula
     */
    public static final String FIND_BY_ID = "Persona_findById";

    /**
     * Constante que identifica la consulta que permite obtener todas las
     * Personas registradas en el sistema
     */
    public static final String GET_ALL = "Persona_getAll";
}
```

Los `NamedQuery` reciben como parámetros dos `String`, el primero de ellos corresponde al nombre de la consulta, el segundo es la consulta como tal. En el ejemplo se han usado constantes para representar los nombres de las consultas, pero también se hubiera podido poner directamente la cadena de caracteres que

representa el nombre de la consulta.

QUERY Y TYPEDQUERY

Para la creación de consultas se puede hacer uso de varios tipos de datos, entre ellos se tienen los `Query` y los `TypedQuery`. Los `Query` hacen uso del tipo `Object` para el retorno de sus resultados, por su parte los `TypedQuery` se basan en `Generics` por lo que los resultados de las consultas se expresan según los parámetros dados en su construcción, ejemplo:

Suponga la entidad `Pais`:

```
@Entity
@NamedQueries({@NamedQuery(name="PAIS_GET_ALL", query="select p from Pais p")})
public class Pais implements Serializable{

}
```

Query:

```
Query query = entityManager.createQuery("select p from Pais p");
List resultados = query.getResultList();
```

O

```
Query query = entityManager.createNamedQuery ("PAIS_GET_ALL");
List resultados = query.getResultList();
```

TypedQuery:

```
TypedQuery<Pais> query = entityManager.createQuery("select p from Pais p", Pais.class);
List<Pais> resultados = query.getResultList();
```

O

```
TypedQuery<Pais> query = entityManager.createNamedQuery ("PAIS_GET_ALL", Pais.class);
List<Pais> resultados = query.getResultList();
```

PRECAUCIONES Y RECOMENDACIONES

Recuerde verificar que el servidor de aplicaciones soporte el motor de base de datos que usará, de igual forma debe verificar que eclipse este haciendo uso del JDK y no del JRE y recuerde adicionar al workspace el servidor de aplicaciones Glassfish antes de crear cualquier proyecto. También puede ser importante verificar que los puertos usados por Glassfish no estén ocupados (Para ello puede hacer uso del comando ***netstat -npl*** o ***netstat -a***)

ARTEFACTOS

Se requiere tener instalado el JDK y un IDE para el desarrollo de aplicaciones (Eclipse JEE en su última

versión), un servidor de aplicaciones que cumpla con las especificaciones de JEE, para esta práctica Glassfish y el motor de base de datos MySQL.

EVALUACIÓN O RESULTADO

Se espera que el alumno pueda usar exitosamente JPA para manipular las entidades y datos de la aplicación.

Procedimiento

1. Para esta guía será necesario hacer uso del proyecto Maven de prueba creado en la guía de pruebas.
2. Cree al menos una entidad, o use entidades creadas previamente. **IMPORTANTE:** No debe olvidar serializar las entidades y sobrescribir el método `equals` en cada una de ellas.
3. El API de persistencia incorpora métodos para la inserción, actualización, borrado y consulta de datos. Sin embargo, en muchas ocasiones se requerirá realizar consultas que no necesariamente harán uso de la llave primaria para la obtención de los datos. Para búsquedas que no se basan en el `id` de la entidad se debe escribir así:

```
Query query = entityManager.createQuery("select entidad from ENTIDAD entidad where  
entidad.atributo=:atributo");  
query.setParameter("atributo", atributo);
```

Si de la búsqueda se espera obtener un único resultado (solo un registro) se puede usar el método `query.getSingleResult()` el cual retorna un objeto, aunque **NO ES RECOMENDADO** usar este método porque si la consulta no arroja resultados o arroja más de un resultado el método arrojará una excepción. Si por el contrario, se espera obtener como resultado de la consulta un conjunto de entidades se debe usar el método `query.getResultList()`, el cual retorna un listado con las entidades que correspondan con la búsqueda, de igual forma el `getResultList()` puede ser usado cuando la consulta arroje solo un resultado, esto es más conveniente dado que `getResultList` no arroja excepción si no se encuentra resultado de la consulta.

Se pueden construir consultas mucho más complejas e incluso usar SQL nativo (Enterprise Java Beans 3.0 5th Edición).

Construya una consulta que le permita obtener la información de todas las personas almacenadas en la base de datos. Haga uso de un `test` que le permita probar el funcionamiento de dicha consulta.

4. Ahora tome la consulta creada en el punto anterior y conviértala en un `NamedQuery`, cree otro método de `test` para comprobar su funcionamiento.
5. Para todas sus entidades cree `NamedQueries` que le permitan obtener todos los registros de dicha entidad. Ejemplo:

```
SELECT entidad FROM Entidad entidad
```

6. Construya una consulta que le permita obtener la persona a la cual corresponden unos datos de autenticación (como por ejemplo email y clave). Haga uso de un `test` que le permita probar el funcionamiento de dicha consulta. De ahora en adelante solo use `NamedQueries`.
7. Obtener todos los registros de una determinada entidad o un subconjunto de estos es muy útil, sin

embargo, esto también puede causar conflictos. Qué pasaría si al realizar una consulta en google en lugar de obtener solo 10 resultados por página google intentará mostrar todos los resultados de la búsqueda en una sola página. Probablemente dejaría de ser el buscador más popular o simplemente no funcionará. Por ellos es importante poder **limitar el número de registros** que arrojan las consultas.

Para limitar el número de registros adicione un parámetro que permita indicar el número máximo de resultados que se desean obtener en el método de consulta realizado. Para lograr la restricción del número de resultados debe hacer uso del método `setMaxResults`, este método hace parte de la consulta (Query) y recibe como parámetro el número máximo de resultados que se desean obtener. Ej.

```
Query query = entityManager.createQuery("select e from ENTIDAD e" );
query = query.setMaxResults(numeroMaximoResultados);
return query.getResultList();
```

Cree un método `test`, donde haga uso de una de las `named queries` creadas para obtener sus registros (No olvide poblar la tabla con suficientes registros para poder observar los resultados) y restrinja el número de los registros obtenidos usando para ello el `setMaxResults`.

8. Si lo que se desea es indicar a partir de qué registro se quiere obtener los resultados se puede hacer uso del método `setFirstResult`. Este método al igual que `setMaxResults` hace parte de la consulta (Query). Por ejemplo, si nuestra consulta arroja 700 resultados, y solo se quieren obtener los últimos 100 (desde el 601 en adelante) se hace lo siguiente:

```
query = query.setFirstResult(601);
return query.getResultList();
```

Cree otro método `test` que ejemplifique el uso de este método.

9. Uso de la cláusula `WHERE`: La cláusula `where` es una de las más importantes, ya que nos permite restringir los resultados según un parámetro dado. Dentro de la cláusula `where` se puede hacer uso de los operadores de navegación (`.`), aritméticos (`+`, `-`, `*`, `/`), de comparación (`=`, `>`, `>=`, `<`, `<=`, `<>` -diferente-, `like`, `between`, `in`, `is null`, `is empty`, `member of`) y lógicos (`not`, `and`, `or`).

Elabore un `NamedQuery` que haga uso de al menos uno de los siguientes operadores: de navegación (`.`), aritméticos (`+`, `-`, `*`, `/`), de comparación (`=`, `>`, `>=`, `<`, `<=`, `<>` -diferente-) y lógicos (`not`, `and`, `or`).

10. Cree un método `test` que le permita comprobar el correcto funcionamiento de su `NamedQuery`.
11. Cree al menos 3 consultas de tipo `NamedQuery` relacionadas a su proyecto y los respectivos métodos de prueba.