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.	09
Nombre de la guía:	Java Persistence Api – JPQL - 2

Información de la Guía

OBJETIVO

Aprender a usar las herramientas que proporciona JPA para la manipulación de las entidades que modelan nuestra aplicación. Hacer uso de JPQL para consultar los datos almacenados y manipular los resultados de las consultas.

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.

La sentencia `SELECT` permite consultar información almacenada en la base de datos. De forma regular el resultado de una consulta suele ser una entidad o un listados de entidades. Sin embargo, no siempre es así, es posible crear consultas en las cuales se obtenga como resultado de una entidad completa sino un subconjunto de sus atributos. En estos casos, el resultado de la consulta será un vector de objetos o una lista de vectores de objetos.

El manejo de vectores de objetos no siempre es cómodo. De hecho, puede presentar problemas en la aplicación tan solo al modificar el orden de los atributos consultados. Es por ello que al hacer este tipo de consulta se recomienda hacer uso de un patrón de diseño llamado `DTO` (Data Transfer Object). Los DTO permiten agrupar información en un solo objeto y de esta forma transmitirla más fácilmente. Básicamente un DTO es una clase compuesta de un conjunto de atributos junto con sus respectivos get y set. Los DTO no poseen lógica.

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 está 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 el desarrollo de esta guía necesitará una base de datos en MySQL, un proyecto de tipo Maven con soporte para el uso de JPA y otro proyecto Maven configurado para la realización de pruebas. Y una conexión a dicha base de datos para ser usada en la generación de las tablas.

2. Para el desarrollo de esta guía haga uso de las entidades creadas para su proyecto. **IMPORTANTE:** No debe olvidar serializar las entidades y sobreescribir el método `equals`.

3. Para los siguientes pasos cree `NamedQueries` y métodos de prueba que le permitan verificar su funcionamiento.

4. La cláusula `SELECT` no solo permite tener acceso a **todos** los atributos de la entidad consultada. También permite tener acceso concretamente a algunos de sus atributos. Por ejemplo, se quiere obtener todos los países que tengan una ciudad con un nombre dado por parámetro.

```
select c.departamento.pais from Ciudad c where c.nombre = :nombreCiudad
```

   Nota: Este tipo de consulta no puede realizarse sobre atributos de tipo `List` (colección de datos) de forma normal. Sin embargo, puede hacerse mediante el uso de cláusulas `IN` o `JOIN`.

   Elabore un `NamedQuery` que dado el ID de un producto devuelva todas sus calificaciones. Escriba un método de tipo test para probar dicha consulta.

5. **Uso del IN**: El `IN` es usado para poder acceder a los elementos de los atributos de tipo `List`. Por ejemplo, si se desea obtener todos los departamentos de Colombia se podría hacer algo como:

```
select d from Pais p, IN (p.departamentos) d where p.codigo = :codigo
```

Elabore un `NamedQuery` que dado el id de una compra, permita obtener todos los productos relacionados a ella (En la entidad Compra). De igual forma elabore un método Test que permita probar la consulta.

6. **Uso del INNER JOIN:** El `INNER JOIN` se puede usar en lugar del `IN`, ambos cumplen la misma función, es así como la cláusula anterior se puede escribir también así: (En los `JOIN` generalmente se usa la cláusula `ON` para limitar la forma en que se combinan las tablas)

```
select d from Pais p INNER JOIN p.departamentos d where p.codigo = :codigo
```

Escriba otro `NamedQuery` que dada la cedula de un usuario se obtenga el listado de compras que ha realizado (En la entidad `Usuario`). De igual forma elabore un método tipo Test que permita probar esta consulta.

7. **Uso del LEFT JOIN (opcional):** El `LEFT JOIN` nos permite hacer consultas incluso cuando nuestra entidad carezca de uno de los atributos requeridos. Por ejemplo, imagine que por algún motivo se quiere obtener el nombre de un curso y los estudiantes inscritos en ellos. Si hacemos uso del `IN` o el `INNER JOIN` podremos realizar la consulta, pero no saldrían en la consulta aquellos cursos que no han tenido inscritos. Para lograr que todos los cursos que aún no tienen registrados estudiantes aparezcan en la consulta se puede hacer uso del `LEFT JOIN`. Al usar el `LEFT JOIN` la consulta incluirá a todos los cursos, incluso aquellos que no tengan aún estudiantes inscritos.

```
select c.nombre, e from Curso c LEFT JOIN c.estudiantes e ON c = e.curso
```

Elabore un `NamedQuery` que permita obtener un listado con la cédula de los usuarios y cada uno de sus productos asociados. El listado debe incluir aquellas personas que no han creado productos. De igual forma debe elaborar un método Test que le permita verificar el correcto funcionamiento de la consulta.

Elabore un `NamedQuery` que permita obtener un listado de los productos y sus calificaciones. El listado debe incluir aquellos productos que no tienen calificaciones. De igual forma debe elaborar un método Test que le permita verificar el correcto funcionamiento de la consulta.

8. **Uso del DISTINCT:** La cláusula `DISTINCT` nos permite eliminar los resultados repetidos en una búsqueda. Por ejemplo, si pensamos en un torneo de fútbol del cual deseamos obtener los jugadores que han anotado goles, podríamos obtener n veces un mismo jugador, donde n es el número de goles que el jugador a anotado en el campeonato. Para evitar eso, debemos usar la cláusula `DISTINCT` así:

```
select DISTINCT g.jugador from Gol g
```

Elabore un `NamedQuery` que permita obtener un listado de los usuarios que han publicado productos (no se deben obtener usuarios repetidos) y un método Test que le permita probar el correcto funcionamiento de esta consulta.

9. Generalmente el resultado de las consultas realizadas son una entidad o un listado de entidades. Sin embargo, no siempre es así. En ocasiones puede requerirse consultar solo parte de la información de una entidad. Para ello, cuando cree el query, después de la cláusula select debe indicar uno a uno los datos que desea obtener. Por ejemplo:

```
select entidad.campo1, entidad.campo2 from ENTIDAD entidad
```

Como los datos no constituyen por sí solos una entidad, el resultado de la consulta no es un listado de entidades como se ha visto. En su lugar contendrá en cada posición de la lista un arreglo de objetos, el cual, a su vez, contendrá en cada una de sus posiciones los elementos indicados en la cláusula select. Recuerde que los elementos de un vector de objetos pueden ser recorridos de diversas formas entre ellas mediante el uso de un iterador.

Elabore un `NamedQuery` que dada una fecha permita obtener un listado con las compras realizadas ese día. Debe retornar el Id de la compra, el método de pago, la cédula y el email de la persona que hizo la compra. De igual forma debe crear un método `Test` que le permita probar el correcto funcionamiento de la consulta.

Elabore un otro `NamedQuery` que devuelva el código y la calificación promedio por cada producto.

Elabore un otro `NamedQuery` que devuelva el código y la calificación promedio por cada producto, incluya aquellos productos que no tienen calificaciones. Escríbala en la entidad Producto.

Escriba otro `NamedQuery` que dado el id de un producto devuelva la calificación promedio que tiene. Cree un método para probar dicha consulta. (use `AVG`)

10. Si se realizan consultas que establecen los campos específicos que se desean obtener y no se desea hacer uso del arreglo de objetos es posible transformar los resultados en un objeto. En este punto cree un `DTO`. El `DTO` es similar a una entidad, solo que sus atributos representan el resultado de nuestra consulta y no es necesario hacer uso de anotaciones. Ejemplo:

```
public class ConsultaDTO{

  private TipoCampo1 campo1;
  private TipoCampo2 campo2;

  public ConsultaDTO(TipoCampo1 campo1,TipoCampo2 campo2){
    this.campo1 = campo1;
    this.campo2 = campo2;
  }
  …
}
```

En la consulta se usaría el `DTO` así:

```
select new co.edu.uniquindio.dto.ConsultaDTO(entidad.campo1,entidad.campo2) from ENTIDAD
entidad
```

Esto nos permitirá obtener un listado de Consultas DTO.

```
@NamedQuery(name="nombreConsulta ", query="select new
co.edu.uniquindio.dto.ConsultaDTO(entidad.campo1,entidad.campo2) from ENTIDAD entidad")
```

Al hacer uso de la consulta se crearía un `TypedQuery` basado en el `DTO` así:

```
TypedQuery<ConsultaDTO> resultados = entityManager.createNamedQuery("nombreConsulta",
ConsultaDTO.class);
```

Cree un método `Test` con base a la consulta del punto anterior, pero en lugar de obtener un listado de arreglos de objetos obtenga un listado de `DTO` que contenga la información solicitada.