



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

Información general	
Actualizado por:	Carlos Andrés Flórez Villarraga
Duración estimada en minutos:	240
Docente:	Christian Andrés Candela y Einer Zapata G
Guía no.	10
Nombre de la guía:	Java Persistence Api – JPQL - 3

Información de la Guía

OBJETIVOS

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.

JPA proporciona herramientas que facilitan la elaboración de consultas complejas algunas de estas herramientas son:

- **MEMBER OF o NOT MEMBER OF:** Permite identificar si una entidad es o no parte de una colección (lista).
- **LIKE:** La cláusula LIKE nos permite condicionar cadenas de caracteres usando un determinado patrón, es decir, nos permite determinar si una cadena cumple con un patrón dado. Ejemplo si deseamos determinar si el nombre de una persona empieza por P tendríamos algo como **nombre LIKE 'P%'**. Si se desea saber si la persona se llama maria, bien sea como nombre inicial o como parte secundaria de su nombre tendríamos algo como **nombre LIKE '%maria%'**. Si adicionalmente no solo queremos las

personas que se llamen maria sino también malia o amalia tendríamos algo como **nombre LIKE '%ma_ia%'**. Ejemplo para obtener una lista de personas cuyo nombre comienza por P:

```
select p from Persona p where p.nombre like 'P%'
```

- **IS EMPTY:** Permite determinar cuándo una colección está vacía. En el siguiente ejemplo se retorna una lista de Personas cuya lista de teléfonos está vacía.

```
select p from Persona p where p.telefonos is empty
```

- **NOT NULL:** nos permite identificar cuando un atributo de la entidad es o no nulo
- **IN:** Permite determinar si un determinado valor se encuentra dentro de un conjunto de valores. En el ejemplo se retorna una lista de ítems cuyo nombre pertenezca a una lista que se pasa por parámetro.

```
select i from Item i where i.nombre IN :nombres
```

- **BETWEEN:** Permite determinar si un valor está dentro de un determinado rango. En el ejemplo se retorna una lista de eventos cuya fecha está entre dos valores que se pasan por parámetro.

```
select e from Eventos e where e.fechaEvento between :fechaInicio and :fechaFin
```

- **ORDER BY:** Permite ordenar los elementos de una consulta por un criterio dado. Si específicamente se desea un orden descendente o ascendente se puede especificar con **ASC** o **DESC**. En el siguiente ejemplo se obtiene una lista de todas las personas ordenadas por la edad de manera descendente:

```
select p from Persona p order by p.edad desc
```

- **GROUP BY:** Esta cláusula es usada para agrupar los resultados de las consultas. En el siguiente ejemplo se agrupan los registros de las compras usando el campo del usuario.

```
select c.usuario from Compra c group by c.usuario
```

- **HAVING:** Esta cláusula es usada en conjunto con el GROUP BY. Nos permite restringir los resultados de la consulta resultante de la agrupación. En el siguiente ejemplo se retornan todos los empleados asignados a 2 o más proyectos

```
select e, count(p) from Empleado e inner join e.proyectos p group by e having count(p)>=2
```

JPA también provee funciones dentro de nuestras consultas. Algunas de las consultas disponibles son:

- **LOWER(string)** convierte la cadena en minúscula
- **UPPER(string)** convierte la cadena en mayúscula
- **TRIM([LEADING | TRAILING | BOTH] [carácter] FROM string)** Permite eliminar caracteres del principio o del final de una cadena (o de ambos). Elimina el carácter que sea indicado, si no se indica ningún carácter a eliminar se asume que es el carácter de espacio. Ejemplo: " pepe " al usar trim

quedaría "pepe".

- **CONCAT**(string,string) pega (concatena) dos cadenas
- **LENGTH**(string) retorna la longitud de la cadena
- **LOCATE**(string1,string2) retorna la posición a partir de la cual se encuentra la cadena 1 en la cadena 2.
- **SUBSTRING**(string,inicio,longitud) retorna una subcadena de la cadena a partir de la posición de inicio y con la longitud indicada.
- **ABS**(número) retorna el valor absoluto del número.
- **SQRT**(numero) retorna el la raíz cuadrada del número.
- **MOD**(numero1,numero2) retorna el residuo de la división del numero1 entre el numero2
- **CURRENT_DATE** Devuelve la fecha actual.

Además de las funciones anteriores que pueden ser usadas en cualquier parte de nuestra consultas, existen otras funciones están diseñadas para ser usadas en la cláusula SELECT como por ejemplo:

- **COUNT** Nos permite determinar la cantidad de registros de una determinada consulta. Por ejemplo, si quisiéramos contar cuántas personas hay por género, sería algo como:

```
select count(p) from Persona p group by p.genero
```

- **MAX** Permite determinar el máximo valor tomado por un campo. En el ejemplo se devuelve la calificación más alta de un espacio académico.

```
select max(r.calificacion) from EspacioAcademico e inner join e.registros r where e.codigo = :codigo
```

- **SUM** Permite sumar los valores de un determinado campo. En el siguiente ejemplo se obtiene la cantidad total de productos que ha comprado un usuario en una determinada compra:

```
select sum(d.cantidad) from DetalleCompra d where d.compra.codigo = :codigo group by d.compra.codigo
```

- **AVG** Permite obtener el promedio de los valores de un determinado campo. En el ejemplo se devuelve el precio promedio de la lista de productos.

```
select avg(p.precio) from Producto p
```

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

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.

Use entidades creadas previamente. **IMPORTANTE:** No debe olvidar serializar las entidades. Para los siguientes pasos cree `NamedQueries` y métodos de prueba que le permitan verificar su funcionamiento.

1. Cree una consulta que permita determinar el número de compras (únicas) que se han realizado. Use COUNT.
2. Cree una consulta que permita contar el número de productos que hay por cada tipo de producto. Use GROUP BY.
3. Cree una consulta que permita determinar qué productos no tienen comentarios. Use IS EMPTY.
4. Cree una consulta que devuelva una lista con todos los usuarios registrados que tienen un email de gmail. Use LIKE.
5. Cree una consulta que permita determinar cuántos productos ha publicado cada usuario. Devuelva un DTO con cédula del usuario, el email y número de registros.
6. Cree una consulta que calcule el valor total que ha gastado un usuario en compras y otra que calcule el valor total que ha ganado un usuario por vender sus productos.
7. Cree una consulta que permita determinar cuál es el tipo de producto que tiene más registros.
8. Cree una consulta que devuelva el producto más costoso que se ha publicado.
9. Cree una consulta que devuelva el producto más costoso por cada tipo de producto.
10. Cree una consulta que devuelva una lista de Compras hechas entre dos fechas determinadas y que además hayan sido pagadas por un método de pago específico.
11. Escriba una consulta que devuelva las compras hechas por una persona, la consulta debe devolver también el detalle de cada compra.

Para la próxima clase

Leer qué significa EJB y para qué se usan.