# JPA - Lenguaje JPQL

El Java Persistence Query Language (JPQL) es el lenguaje estándar de consultas de JPA. Es un lenguaje diseñado para combinar la simplicidad de la semántica y sintaxis del lenguaje SQL con la expresividad de un lenguaje orientado a objetos.

# Introducción:

JPQL no es SQL. A pesar de las similitudes en los dos lenguajes en términos de estructura y palabras clave hay diferencias importantes.

Las similitudes en los lenguajes son intencionadas ya que permite a los desarrolladores con conocimientos de SQL que la curva de aprendizaje sea menor, sin embargo, la naturaleza orientada a objetos del lenguaje JPQL requiere un manera de pensar diferente.

JPQL es un lenguaje de consulta de entidades/objetos en lugar de tablas y registros. El lenguaje nos proporciona una manera de expresar consultas en términos de entidades y sus relaciones operando sobre las entidades en lugar de sobre el modelo de BBDD.

JPQL es portable, es decir, puede ser traducido al dialecto SQL utilizado por la mayoría de BBDD

JPQL permite abstraernos de la BBDD, es decir, las consultas son escritas sobre el modelo de entidades/objetos sin necesidad de conocer como estas son mapeadas a la BBDD.

# Tipos de Sentencias:

Sobre cada tipo haremos hincapié en las diferencias con el lenguaje SQL, asumiendo que el lector ya tiene conocimientos de este:

1. SENTENCIAS SELECT
2. SENTENCIAS AGGREGATE
3. SENTENCIAS UPDATE
4. SENTENCIAS DELETE

# SENTENCIAS SELECT:

Son las sentencias más significativas, y permiten la recuperación de un conjunto de entidades de la BBDD. Su estructura es la siguiente:

```
SELECT e
FROM Employee e
```

El dominio definido en la cláusula FROM no es una tabla sino una entidad

### CLÁUSULA SELECT

La cláusula SELECT no enumera los campos o utiliza un "*" para seleccionar todos los campos. En su lugar, utiliza la variable "e" que indica que el tipo de resultado es una entidad *Employee*

```
SELECT e.department
FROM Employee e
```

La cláusula SELECT utiliza una expresión de ruta para entidad *Deparment*

```
SELECT e.name, e.salary
FROM Employee e
```

La cláusula SELECT utiliza múltiples expresiones que determinan que el tipo de resultado es un *Object* con dos elementos, el primero un *String* asociado al name y un *Long* asociado al salary

```
SELECT NEW example.EmployeeDetails(e.name, e.salary, e.department.name)
FROM Employee e
```

La cláusula SELECT utiliza una expresión de construcción que define que el tipo de resultado es una entidad de la clase example.EmployeeDetails. El motor de consulta buscará un constructor de la clase que corresponda con los 3 parámetros utilizados en la expresión (*String*, *Long*, *String*)

### CLÁUSULA FROM

Se usa para declarar una o más variables de identificación. Las variables de identificación son el punto de entrada para las expresiones de consultas . Debe existir al menos una variable de identificación definida en la cláusula FROM

*Nota*: Cuando una declaración de variable de identificación es únicamente un nombre de Entidad se conoce como declaración de variable de rango.

**JOIN**

Un Join es una consulta que combina resultados de distintas entidades. Cuando la consulta es traducida a SQL es común que las join entre entidades produzcan join similares entre las tablas

**Inner Join**

Un inner join devuelve todas las entidades del lado izquierdo que tienen una relación con la entidad del lado derecho de la relación

Se pueden definir de forma explícita utilizando el operador JOIN en la cláusula FROM

```
SELECT p
FROM Employee e JOIN e.phones p
```

Su traducción a SQL será:

```
SELECT p.id, p.phone_num, p.type, p.emp_id
FROM emp e, phone p
WHERE e.id = p.emp_id
```

Otra manera es mediante una variable de identificación en la cláusula FROM y condiciones de unión sobre ella en la cláusula WHERE. Este tipo de consulta se utiliza cuando no hay una relación explícita entre dos entidades en el modelo de dominio

```
SELECT DISTINCT d
FROM Department d, Employee e
WHERE d = e.department
```

**Outer Join**

Un outer join devuelve todas las entidades del lado izquierdo de la relación y la entidad de lado derecho con la que se relaciona

```
SELECT e, d
FROM Employee e LEFT JOIN e.department d
```

Su traducción a SQL será:

```
SELECT e.id, e.name, e.salary, e.manager_id, e.dept_id, e.address_id,
d.id, d.name
FROM employee e LEFT OUTER JOIN department d
ON (d.id = e.department_id)
```

En el ejemplo anterior sino existe departamento el segundo objeto del Objeto devuelto será null

**Fetch Join**

Permite seleccionar una o varias relaciones que deben ser traídas de forma “ansiosa”, es decir, modificar el comportamiento por defecto que lo que hace es cargar las entidades relacionadas de forma “perezosa”

```
SELECT e
FROM Employee e JOIN FETCH e.address
```

#### CLÁUSULA WHERE

Se utiliza para añadir condiciones de filtrado y reducir el número de resultados de la consulta. Veremos a continuación ejemplos de las expresiones condicionales más importantes.

BETWEEN permite filtrar por un rango de valores inicial y final (incluidos) de
tipo *Numeric*, *String* o *Date*

```
SELECT e
FROM Employee e
WHERE e.salary BETWEEN 40000 AND 45000
```

LIKE permite filtrar por un patrón. El carácter “%” representa cualquier carácter o caracteres

```
SELECT d
FROM Department d
WHERE d.name LIKE '__Eng%'
```

IN permite filtrar que un valor único está dentro de una lista de valores. En este caso el estado debe ser NY o CA

```
SELECT e
FROM Employee e
WHERE e.address.state IN ('NY', 'CA')
```

IS EMPTY o IS NOT EMPTY filtra por colecciones vacías o viceversa

```
SELECT e
FROM Employee e
WHERE e.directs IS NOT EMPTY
```

EXISTS o NOT EXISTS permite filtra por subconsultas que devuelven algún valor

```
SELECT e
FROM Employee e
WHERE NOT EXISTS (SELECT p
FROM e.phones p
WHERE p.type = 'Cell')
```

ANY, ALL permite filtrar por subconsultas para indicar que alguna entidad debe cumplir la condición o todas deben cumplirla

```
SELECT e
FROM Employee e
WHERE e.directs IS NOT EMPTY AND
e.salary < ALL (SELECT d.salary
FROM e.directs d)
```

#### CLÁUSULA ORDER

Permiten especificar el orden de los resultados. Las palabras clave ASC y DESC son opcionales y colocadas a continuación de la expresión de ordenamiento indican el sentido de la ordenación (ascendente o descendente). Por defecto sino se indica nada el orden es ascendente

```
SELECT e, d
FROM Employee e JOIN e.department d
ORDER BY d.name, e.name DESC
```

# SENTENCIAS AGREGADAS

#### CLÁUSULA SELECT

Se pueden utilizar 5 funciones agregadas en la cláusula SELECT AVG, COUNT, MAX, MIN, and SUM

**AVG** Calcula la media del valor de un campo sobre el grupo. El tipo del campo debe ser numérico y el resultado será un Double

**COUNT** sirve para contar cuantos elementos hay sobre el grupo. Puede añadirse la palabra reservada DISTINCT para que elimine los valores duplicados antes de contar

```
SELECT e, COUNT(p), COUNT(DISTINCT p.type)
FROM Employee e JOIN e.phones p
GROUP BY e
```

**MAX** Calcula el valor máximo de un campo sobre el grupo

**MIN** Calcula el valor mínimo de un campo sobre el grupo
**SUM** Calcula la suma de valores de un campo sobre el grupo

**CLÁUSULA GROUP BY**
Define la expresión de agrupación sobre la cual los resultados serán agregados.
Debe ser una expresión de valor único como un campo o una entidad

```
SELECT d.name, COUNT(e)
FROM Department d JOIN d.employees e
GROUP BY d.name
```

Se pueden aplicar varias agregaciones en la misma consulta

```
SELECT d.name, COUNT(e), AVG(e.salary)
FROM Department d JOIN d.employees e
GROUP BY d.name
```

**CLÁUSULA HAVING**

Define un filtro que será aplicado después de que los resultados de la consulta han sido agrupados.
Se puede considerar como una segunda cláusula de filtro que permite a su vez el uso de funciones agregadas

```
SELECT e, COUNT(p)
FROM Employee e JOIN e.projects p
GROUP BY e
HAVING COUNT(p) >= 2
```

El ejemplo recupera todos los empleados asignados a 2 o más proyectos

# SENTENCIAS UPDATE

Son equivalentes a los SQL Update y las usaremos para actualizar las propiedades de nuestras entidades.
La cláusula WHERE es equivalente a la de las consultas SELECT
Se pueden actualizar varias propiedades en una única sentencia
Su estructura es la siguiente:

```
UPDATE entity_name [[AS] identification_variable]
SET update_statement {, update_statement}*
[WHERE conditional_expression]
```

```
UPDATE Phone p
SET p.number = CONCAT('288', SUBSTRING(p.number, LOCATE(p.number, '-'), 4)),
p.type = 'Business'
WHERE p.employee.address.city = 'Ottawa' AND
p.type = 'Office'
```

La anterior sentencia modifica los números de teléfono y tipo de los empleados de la ciudad de Otawa

# SENTENCIAS DELETE

Son equivalentes a los SQL DELETE y las usaremos para eliminar entidades. Su estructura es la siguiente:

DELETE FROM entity_name [[AS] identification_variable]
[WHERE condition]

```
DELETE FROM Employee e
WHERE e.department IS NULL
```

La cláusula WHERE es equivalente a la de las consultas SELECT. Sino se especifica todas las entidades del tipo indicado serán eliminadas