**Consultas SELECT**

Las consultas SELECT son el tipo de consulta más significativo ya que permiten una recuperación masiva de datos de la base de datos. La estructura general de una SELECT es la siguiente:

SELECT <select\_expression>

FROM <from\_clause>

[WHERE <conditional\_expression>]

[ORDER BY <order\_by\_clause>]

Una consulta SELECT, tiene dos partes obligatorias: la cláusula SELECT que define el formato de los resultados de consulta y la cláusula FROM que define la entidad o entidades de las que provendrán los resultados. A continuación, se va a ver un ejemplo de una consulta SELECT que devuelve todos los empleados:

SELECT e

FROM Employee e

La estructura de esta consulta es muy similar a SQL con la diferencia que la cláusula FROM hace referencia a entidades, no tablas. Además, a diferencia de las consultas de SQL, donde el alias es opcional, en JP QL es obligatorio.

La segunda diferencia, es que se utiliza el alias de la entidad para poder seleccionar todos los campos de esta.

**Cláusula SELECT**

Expresiones de ruta

Las expresiones de ruta son los bloques de construcción de las consultas. Se utilizan para navegar desde una entidad, ya sea a través de una relación con otra entidad o con una de las propiedades persistentes de la entidad.

El operador punto(.) significa navegación de trayectoria en una expresión. De esta manera, si la entidad Empleado se le ha asignado la variable de identificación e, e.name, es la expresión que nos devolvería el nombre del empleado.

Lo que hace que las expresiones de ruta sean tan poderosas es que no se limitan a una sola navegación. De esta manera, se puede construir expresiones de ruta como

e.department.name, que es el nombre del departamento al que pertenece el empleado.

Entidades y Objetos

La estructura más simple de la cláusula SELECT es una única variable de identificación. Lo que devuelve la entidad completa a la cual está asociada esta variable. Por ejemplo, la consulta siguiente devolvería todos los departamentos de la empresa:

SELECT d

FROM Department d

La palabra clave OBJECT puede ser usada para resaltar, visualmente, que lo que se está devolviendo es un objeto entero. Es decir la expresión OBJECT(e.department) no sería válida ya que es una propiedad del objeto empleado, no el objeto en cuestión. A efectos prácticos la consulta anterior y la siguiente devuelven el mismo resultado:

SELECT OBJECT(d)

FROM Department d

También, es posible utilizar la cláusula SELECT para devolver un único campo de una entidad. La siguiente consulta devuelve los nombres de todos los empleados:

SELECT e.name

FROM Employee e

El resultado de esta consulta es un conjunto de String, por lo que la ejecución de esta consulta usando getResultList() devolverá una colección de cero o más String.

La cláusula SELECT no solo devuelve propiedades de una única entidad, sino que también es posible devolver una entidad diferente por medio de una navegación por ruta. La siguiente consulta muestra como devolver una entidad diferente como resultado de una navegación por ruta:

SELECT e.department

FROM Employee e

El resultado de esta consulta es el nombre del departamento al que pertenece cada empleado, ya que es el resultado de atravesar la relación del departamento entre Empleado y Departamento.

Para remover los duplicados, se añade el operador DISTINCT.

SELECT DISTINCT e.department

FROM Employee e

El operador DISTINCT funciona prácticamente igual que el de SQL. Una vez se ha recogido el conjunto de resultados de una consulta, se eliminan los valores duplicados.

También es posible seleccionar objetos incrustables a los que se ha navegado en una expresión de ruta. La siguiente consulta devuelve los objetos incrustables ContactInfo para todos los empleados:

SELECT e.contactInfo

FROM Employee e

Lo único que hay que tener en cuenta al seleccionar los objetos incrustados es que los objetos no se gestionarán. Si se realiza una consulta para devolver a los empleados y, a continuación, desde los resultados, se navega hasta los objetos incrustados ContactInfo, se obtendrán todos los objetos incrustados que se hayan gestionado. la modificación de cualquiera de estos objetos se grabará una vez se confirme la operación.

Combinando Expresiones

Se pueden especificar múltiples expresiones en la misma cláusula SELECT separándolas con comas. El tipo de resultado de la consulta en este caso es una matriz de tipo objeto, donde los elementos de la matriz son los resultados de resolver las expresiones en el orden en que aparecieron en la consulta.

Considere la siguiente consulta que sólo devuelve el nombre y el salario de un empleado:

SELECT e.name, e.salary

FROM Employee e

Cuando esto se ejecuta, se devuelve una colección de cero o más instancia de matrices del tipo Objeto. Cada array en este ejemplo tiene dos elementos, el primero es una cadena que contiene el nombre del empleado y el segundo es un doble que contiene el salario del empleado.

Este tipo de consultas suele hacerse para ahorrar un esfuerzo adicional a la hora de construir una instancia de la entidad. Por ello, cuando solo se quiere devolver un conjunto mínimo de datos, es más útil utilizar este tipo de consultas que las que devuelven la entidad completa.

Expresiones Constructoras

Una forma más potente de la cláusula SELECT es almacenar los datos devueltos por la consulta en un tipo de objeto especificado por el usuario.

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

FROM Employee e

El resultado de esta consulta es una clase de Java (example.EmployeeDetails). Como el procesador de consultas itera sobre los resultados de la consulta, utilizará un constructor que coincida con los tipos de expresión listados en la sección consulta. En este caso, los tipos de expresión son String, Double, String, por los que la consulta buscará un constructor con esos tipos para los argumentos.

Se debe hacer referencia al tipo de objeto de resultado empleando el nombre totalmente cualificado de la clase. Sin embargo, la clase no tiene que ser asignada a la base de datos. Cualquier constructor compatible con las expresiones listadas en la cláusula SELECT puede ser usado en una expresión constructiva.

**Cláusula FROM**

La cláusula FROM se utiliza para declarar una o más variables de identificación. Opcionalmente, no sólo buscará en la variable de identificación que se le ha declarado, sino también en todas las que dependen de ella. La sintaxis de la cláusula FROM consiste en una o más variables de identificación y declaraciones de cláusulas de unión.

La variable de identificación es el punto de partida de todas las expresiones de consulta. Cada consulta debe tener al menos una variable de identificación definida en la cláusula FROM, y esa variable debe corresponder a un tipo de entidad. Cuando una declaración de variables de identificación no utiliza una expresión de ruta, se denomina declaración de variables de rango.

las declaraciones de variables de rango utilizan la sintaxis <nombre\_de\_entidad<[AS] <identificador>. Se ha estado usando esta sintaxis en ejemplos anteriores, pero sin la necesidad de utilizar la palabra clave AS. El identificador debe seguir las reglas de nomenclatura estándar de Java y puede ser referenciado a lo largo de la consulta de forma insensible a mayúsculas y minúsculas. Se pueden especificar varias declaraciones separándolas con comas.

Las expresiones de ruta también se pueden aliar a variables de identificación en el caso de uniones y subconsultas.

**JOINS**

Un Join es una consulta que combina resultados de múltiples entidades. Las uniones en las consultas JP QL son equivalentes a las uniones en SQL. En última instancia, una vez que la consulta se traduce a SQL, es muy probable que las uniones entre entidades produzcan uniones similares entre las tablas a las que se asignan. Por lo tanto, es importante comprender cuándo se producen las uniones para escribir consultas eficaces.

Las uniones se producen cuando se cumple alguna de las siguientes condiciones en una consulta SELECT:

* En la cláusula FROM se enumeran dos o más declaraciones de variables de rango y aparecen en la cláusula SELECT.
* El operador JOIN se utiliza para ampliar una variable de identificación utilizando una expresión de ruta.
* Una expresión de ruta en cualquier parte de la consulta navega a través de un campo de asociación, hala misma o hacia una entidad diferente.
* Una o más condiciones WHERE comparan atributos de diferentes variables de identificación.

Una unión interna entre dos entidades devuelve los objetos de ambos tipos de entidades que cumplen todas las condiciones de unión. la ruta de navegación de una entidad a otra es una forma de unión interna. la unión externa de dos entidades es el conjunto de objetos de ambos tipos de entidades que satisfaces las condiciones de unión más el conjunto de objetos de un tipo de entidad que no tienen ninguna condición de unión correspondiente en la otra.

En ausencia de condiciones de unión entre dos entidades, las consultas producirán un producto cartesiano entre las dos entidades. Los productos cartesianos son raros en las consultas JP QL dadas las capacidades de navegación del idioma, pero son posibles si se especifican dos declaraciones de variables de rango en la cláusula FROM sin condiciones adicionales especificadas en la cláusula WHERE.

A continuación, se verán con más detalle cada tipo de unión.

**INNER JOIN (Unión interna)**

Todas las consultas anteriores han utilizado la cláusula FROM poniéndole un alias a la variable de identificación. Sin embargo, como lenguaje relacional, JP QL soporta consultas que se basan en múltiples entidades y las relaciones entre ellas.

La forma típica de las uniones internas entre dos entidades es el uso del operador JOIN en la cláusula FROM.

Campos del operador JOIN y Asociación de recogida.

La sintaxis de una unión interna utilizando el operador JOIN es: [INNER] JOIN <path\_expression>[AS] <identificador>.

SELECT p

FROM Employee e JOIN e.phones p

Esta consulta utiliza el operador JOIN para unir la entidad del empleado con la entidad del teléfono. la condición de unión en esta consulta está definida por el mapeo objeto-relacional de la relación entre los teléfonos. No es necesario especificar criterios adicionales para vincular a las dos entidades. al unir las dos entidades, esta consulta devuelve todas las instancias de entidades telefónicas asociadas con los empleados de la empresa.

La sintaxis de la unión es similar a la de las expresiones JOIN soportadas por ANSI SQL. También es posible realizar la misma consulta en su forma tradicional de unión:

SELECT p.id, p.phone\_num, p.type, p.emp\_id

FROM emp e, phone p

WHERE e.id = p.emp\_id

El mapeo de la tabla Teléfono reemplaza la expresión e.phones. La cláusula WHERE también incluye los criterios necesarios para unir las dos tablas a través de la unión definida por el mapeo de teléfonos.

El uso de JOIN, permite evitar que Java tenga que acceder a los campos de una entidad por medio otra. De tal manera que, mediante la unión de las dos tablas, podemos acceder a todos los campos de la entidad TELÉFONO simplemente navegando por la tabla de unión.

SELECT p.number

FROM Employee e JOIN e.phones p

Para evitar que una ruta no pueda continuar desde un campo de estado o un campo de asociación de colección, el campo de asociación de la colección debe estar unido en la cláusula FROM de forma que se cree una nueva variable de identificación para la ruta.

IN versus JOIN

La consulta vista anteriormente, se puede expresar mediante la cláusula IN, devolviendo el mismo resultado

SELECT DISTINCT p

FROM Employee e, IN(e.phones) p

El operador IN tiene por objeto indicar que la variable p es una enumeración de la colección de teléfonos. En cambio, el operador JOIN es una forma más poderosa y expresiva de declarar relaciones, además de ser el operador recomendado para las consultas.

Operador JOIN y campos de Asociación de valor individual

El operador JOIN trabaja tanto con expresiones de ruta de asociación de valor de colección como con expresiones de ruta de asociación de valor único

SELECT d

FROM Employee e JOIN e.department d

Esta consulta define una unión de Empleado a Departamento a través de la relación de departamento. Esto es semánticamente equivalente a usar una expresión de ruta en la cláusula SELECT para obtener el departamento para el empleado.

La posibilidad de uniones internas implícitas resultantes de las expresiones de ruta es algo de lo que hay que ser consciente. Considere el siguiente ejemplo que devuelve los distintos departamentos situados en California que están participando en el proyecto Release1:

SELECT DISTINCT e.department

FROM Project p JOIN p.employees e

WHERE p.name = 'Release1' AND

e.address.state = 'CA'

En realidad, hay cuatro uniones lógicas aquí, no dos. El traductor tratará la consulta como si hubiera sido escrita con uniones explícitas entre las distintas entidades. La consulta siguiente es equivalente a la anterior.

SELECT DISTINCT d

FROM Project p JOIN p.employees e JOIN e.department d JOIN e.address a

WHERE p.name = 'Release1' AND

a.state = 'CA'

Decimos cuatro uniones lógicas porque el mapeo físico real podría involucrar más tablas. En este caso, el empleado y las entidades del proyecto se relacionan a través de una asociación de muchos a muchos utilizando una tabla de unión. Por lo tanto, el SQL real para tal consulta utiliza cinco tablas, no cuatro.

SELECT DISTINCT d.id, d.name

FROM project p, emp\_projects ep, emp e, dept d, address a

WHERE p.id = ep.project\_id AND

ep.emp\_id = e.id AND

e.dept\_id = d.id AND

e.address\_id = a.id AND

p.name = 'Release1' AND

a.state = 'CA'

Condiciones de unión en la cláusula WHERE

Las consultas de SQL han unido, tradicionalmente, las tablas mediante la enumeración de las tablas que deben unirse en la cláusula FROM y los filtros necesarios en la cláusula WHERE.

El ejemplo anterior de la unión entre el Empleado y la entidad Departamento también podría haberse escrito así.

SELECT DISTINCT d

FROM Department d, Employee e

WHERE d = e.department

Este estilo de consulta se utiliza normalmente para compensar la falta de una relación explícita entre dos entidades en el modelo de dominio.

Se puede utilizar una condición de unión en la cláusula WHERE para que esto sea posible.

SELECT d, m

FROM Department d, Employee m

WHERE d = m.department AND

m.directs IS NOT EMPTY

En el ejemplo anterior, se utiliza la expresión IS NOT EMPTY que verifica que la colección *directs* no está vacía y en caso de no estarlo estaríamos antes un gerente.

Uniones múltiples

En algunos casos es necesario unir en cascada. Por ejemplo, la consulta siguiente devuelve el conjunto de proyectos de un empleado que pertenece a un departamento.

SELECT DISTINCT p

FROM Department d JOIN d.employees e JOIN e.projects p

El procesador de consultas interpresa la cláusula FROM de izquierda a derecha, de tal manera, que es posible hacer referencia a una variable declara anteriormente en otras expresiones JOIN.

**MAP JOINS (mapa de Uniones)**

Una expresión de ruta que navega a través de una asociación de valor de colección implementado como un Mapa es un caso especial. A diferencia de una colección normal, cada elemento de un mapa correspondo a dos piezas de información: la clave y el valor. Cuando se trabaja con JP QL es importante señalar que las variables de identificación basadas en mapas se refieren al valor por defecto. Por ejemplo, considérese el caso en el que la relación telefónica del Empleado se modela como un mapa, donde la clave es el tipo de número y el valor es el número de teléfono.

SELECT e.name, p

FROM Employee e JOIN e.phones p

Este comportamiento puede ser resaltado explícitamente mediante el uso de la palabra clave VALUE.

SELECT e.name, VALUE(p)

FROM Employee e JOIN e.phones p

Para acceder a la clave en lugar del valor de un elemento de mapa determinado, podemos utilizar la palabra clave KEY para anular el comportamiento predeterminado y devolver el valor de la clave para un elemento de mapa determinado. El siguiente ejemplo muestra cómo añadir el tipo de teléfono a la consulta anterior:

SELECT e.name, KEY(p), VALUE(p)

FROM Employee e JOIN e.phones p

WHERE KEY(p) IN ('Work', 'Cell')

Finalmente, en el caso de que queramos que la clave y el valor se devuelvan juntos en forma de un objeto java.util.map.entry, podemos especificar la palabra clave ENTRY de la misma manera. Tenga en cuenta que la palabra clave ENTRY sólo se puede utilizar en la cláusula SELECT. Las palabras clave KEY y VALUE también se pueden utilizar como parte de expresiones condicionales en las cláusulas WHERE y HAVING de la consulta.

Hay que tener en cuenta que, en cada uno de los ejemplos de unión de mapa, unimos una entidad contra uno de sus atributos de mapa y obtuvimos un par clave, valor o valor clave (entry). Sin embargo, cuando se ve desde la perspectiva de las tablas, la unión sólo se hace a nivel de la clave primaria de la entidad de origen y de los valores del mapa. Actualmente no hay ninguna instalación disponible en JPA para unirse a la entidad fuente contra las claves del Mapa.

**OUTER JOIN (Unión externa)**

Un outer join entre dos entidades produce un dominio en el que sólo se requiere que un lado de la relación esté completo. Es decir, la unión externa de Empleado a Departamente mediante el campo departamento de empleado, devuelve a todos los empleados y el departamento al cual estén asignados, exista o no ese departamento, pero solo si el departamento tiene asignado un empleado este será devuelto.

Se especifica una unión externa utilizando la siguiente sintaxis: LEFT[OUTER] JOIN <path\_path expresión>[AS]<identificador>. La siguiente consulta muestra la unión externa entre dos entidades:

SELECT e, d

FROM Employee e LEFT JOIN e.department d

Si el empleado no ha sido asignado a un departamento, el objeto departamento será nulo.

En una generación típica de SQL el código anterior equivale a lo siguiente:

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)

El SQL resultante muestra que cuando se genera una unión externa a partir de JP QL siempre especifica una condición ON de igualdad entre la columna de unión que mapea la relación a través de la cual se está uniendo y la clave primaria a la que se está haciendo referencia.

Se puede suministrar una expresión ON adicional para añadir restricciones a los objetos que obtienen devuelto desde el lado derecho de la unión. Por ejemplo, podemos modificar el JP QL anterior la consulta para tener una condición adicional ON para limitar los departamentos devueltos sólo a aquellos que tienen un prefijo `QA':

SELECT e, d

FROM Employee e LEFT JOIN e.department d

ON d.name LIKE 'QA%'

Esta consulta todavía devuelve a todos los empleados, pero los resultados no incluirán ningún departamento que no coincida con la condición ON añadida. El SQL generado tendría el siguiente aspecto esto:

SELECT e.id, e.name, e.salary, e.department\_id, e.manager\_id, e.address\_id,

d.id, d.name

FROM employee e left outer join department d

ON ((d.id = e.department\_id) and (d.name like 'QA%'))

Destacar que esta consulta es muy diferente de usar una expresión WHERE ya que la cláusula WHERE provoca un inner join entre el Empleado y el Departamento, por lo que esta consulta sólo devolvería a los empleados que estaban en un departamento con un nombre prefijado'QA'.

SELECT e, d

FROM Employee e LEFT JOIN e.department d

WHERE d.name LIKE 'QA%'

**FECTH JOINS**

Los Fetch joins están pensados para ayudar a los diseñadores de aplicaciones a optimizar el acceso a la base de datos y a preparar los resultados de las consultas. Permiten que en las consultas se especifique una o más relaciones que deben ser navegadas y preprogramadas por el motor de consultas para que no tarden en cargar en tiempo de ejecución.

Se debe tener en cuenta que no se ha definido ninguna variable de identificación para la expresión de la ruta de la dirección e. Esto se debe a que, aunque la entidad Address se está uniendo para resolver la relación, no forma parte del tipo de resultado de la consulta. El resultado de ejecutar la consulta sigue siendo una colección de instancias de entidades de Empleados, con la excepción que la relación de direcciones en cada entidad no causará un segundo viaje a la base de datos cuando se acceda a ella. Esto también permite acceder a la relación de direcciones de forma segura si la entidad del empleado desaparece. Un Fetch Join se distingue de un Join regular añadiendo la palabra clave FETCH al operador JOIN.

Para implementar Fetch Joins, el proveedor necesita convertir la asociación de búsqueda en una unión regular del tipo apropiado: interna por defecto o externa si se especificó la palabra clave LEFT. La expresión SELECT de la consulta también necesita ser expandida para incluir la relación Join. Expresado en JP QL, el Fetch Join de antes se vería así:

SELECT e, a

FROM Employee e JOIN e.address a

La única diferencia es que el proveedor no devuelve las entidades de dirección a la persona que llama. Debido a que los resultados se procesan a partir de esta consulta, el motor de consulta crea la entidad de dirección en memoria y la asigna a la entidad de empleado, pero luego la elimina de la colección de resultados que crea para el cliente. Esto cara la relación de direcciones a la que se puede acceder navegando desde la entidad Empleado.

Una consecuencia de la implementación de los Fetch Joins de esta manera es que la obtención de una asociación de colección da como resultado resultados duplicados. Por ejemplo, considere una consulta del departamento donde hay una gran relación de los empleados de la entidad del Departamento. La consulta Fetch Join, esta vez utilizando un enlace externo para asegurar que se recuperan los departamentos sin empleados, se escribiría de la siguiente manera:

SELECT d

FROM Department d LEFT JOIN FETCH d.employees

En cambio, si se realiza una unión externa entre las dos tablas, se pueden ver todos los empleados que pertenecen a cada departamento.

SELECT d, e

FROM Department d LEFT JOIN d.employees e

Cada entidad del Departamento tiene ahora una colección de empleados totalmente completa, pero el cliente recibiría una referencia a cada departamento por empleado. Es decir, si se recuperaran cuatro departamentos con cinco empleados cada uno, el resultado sería una colección de 20 instancia del Departamento, con cada departamento duplicado cinco veces. Para evitar esta repetición los valores duplicados se debe utilizar el operador DISTINCT.

Dados los resultados anteriores, realizar una unión de una colección a otra puede que no sea la forma más apropiada de cargar entidades relacionadas en todos los casos. si una colección requiere que se busque de forma regular, se debe considerar dicha posibilidad. (esto Algunos proveedores de persistencia también ofrecen lecturas de lotes como una alternativa para obtener uniones que emiten múltiples consultas en un solo lote y luego correlacionan los resultados con relaciones de dependencia. otra alternativa es utilizar un gráfico de entidad para determinar dinámicamente los atributos de la relación que se van a cargar mediante una consulta.