Anterior Siguiente

# Capítulo 15. HQL: El lenguaje de consulta de Hibernate



Hibernate utiliza un lenguaje de consulta potente (HQL) que se parece a SQL. Sin embargo, comparado con SQL, HQL es completamente orientado a objetos y comprende nociones como herencia, polimorfismo y asociación.

## 15.1. Sensibilidad a mayúsculas

Las consultas no son sensibles a mayúsculas, a excepción de los nombres de las clases y propiedades Java. De modo que SeLeCT es lo mismo que sELEct e igual a SELECT, pero org.hibernate.eg.FOO no es lo mismo que org.hibernate.eg.Foo y foo.barSet no es igual a foo.BARSET.

Este manual utiliza palabras clave HQL en minúsculas. Algunos usuarios encuentran que las consultas con palabras clave en mayúsculas son más fáciles de leer, pero esta convención no es apropiada para las peticiones incluidas en código Java.

## 15.2. La cláusula from

La consulta posible más simple de Hibernate es de esta manera:

```
from eg.Cat
```

Esto retorna todas las instancias de la clase eg.Cat. Usualmente no es necesario calificar el nombre de la clase ya que auto-import es el valor predeterminado. Por ejemplo:

```
from Cat
```

Con el fin de referirse al Cat en otras partes de la petición, necesitará asignar un *alias*. Por ejemplo:

```
from Cat as cat
```

Esta consulta asigna el alias cat a las instancias Cat, de modo que puede utilizar ese alias luego en la consulta. La palabra clave as es opcional. También podría escribir:

```
from Cat cat
```

Pueden aparecer múltiples clases, lo que causa un producto cartesiano o una unión "cruzada" (cross join).

```
from Formula, Parameter
```

```
from Formula as form, Parameter as param
```

Se considera como una buena práctica el nombrar los alias de consulta utilizando una inicial en minúsculas, consistente con los estándares de nombrado de Java para las variables locales (por ejemplo, domesticCat).

## 15.3. Asociaciones y uniones (joins)

También puede asignar alias a entidades asociadas o a elementos de una colección de valores utilizando una join. Por ejemplo:

```
from Cat as cat
    inner join cat.mate as mate
    left outer join cat.kittens as kitten
```

```
from Cat as cat left join cat.mate.kittens as kittens
```

```
from Formula form full join form.parameter param
```

Los tipos de uniones soportadas se tomaron prestados de ANSI SQL

- inner join
- left outer join
- right outer join
- full join (no es útil usualmente)

Las construcciones inner join, left outer join y right outer join se pueden abreviar.

```
from Cat as cat
    join cat.mate as mate
    left join cat.kittens as kitten
```

Puede proveer condiciones extras de unión utilizando la palabra clave with de HQL.

```
from Cat as cat
    left join cat.kittens as kitten
        with kitten.bodyWeight
> 10.0
```

A "fetch" join allows associations or collections of values to be initialized along with their parent objects using a single select. This is particularly useful in the case of a collection. It effectively overrides the outer join and lazy declarations of the mapping file for associations and collections. See Sección 20.1, “Estrategias de recuperación” for more information.

```
from Cat as cat
    inner join fetch cat.mate
    left join fetch cat.kittens
```

Usualmente no se necesita asignársele un alias a una unión de recuperación ya que los objetos asociados no se deben utilizar en la cláusula where (ni en cualquier otra cláusula). Los objetos asociados no se retornan directamente en los resultados de la consulta. En cambio, se pueden acceder por medio del objeto padre. La única razón por la que necesitaríamos un alias es si estamos uniendo recursivamente otra colección:

```
from Cat as cat
    inner join fetch cat.mate
    left join fetch cat.kittens child
    left join fetch child.kittens
```

La construcción fetch no puede utilizarse en consultas llamadas que usen iterate() (aunque se puede utilizar scroll()). Fetch se debe usar junto con setMaxResults() o setFirstResult() ya que estas operaciones se basan en las filas de resultados, las cuales usualmente contienen duplicados para la recuperación de colección temprana, por lo tanto, el número de filas no es lo que se esperaría. Fetch no se debe usar junto con una condición with improvisadas. Es posible crear un producto cartesiano por medio de una recuperación por union más de una colección en una consulta, así que tenga cuidado en este caso. La recuperación por unión de múltiples roles de colección también da resultados a veces inesperados para mapeos de bag, así que tenga cuidado de cómo formular sus consultas en este caso. Finalmente, observe que full join fetch y right join fetch no son significativos.

Si está utilizando una recuperación perezosa a nivel de propiedad (con instrumentación de código byte), es posible forzar a Hibernate a traer las propiedades perezosas inmediatamente utilizando fetch all properties.

```
from Document fetch all properties order by name
```

```
from Document doc fetch all properties where lower(doc.name) like '%cats%'
```

## 15.4. Formas de sintaxis unida

HQL soporta dos formas de unión de asociación: implicit y explicit.

Las consultas que se mostraron en la sección anterior todas utilizan la forma explicit, en donde la palabra clave join se utiliza explícitamente en la claúsula from. Esta es la forma recomendada.

La forma implicit no utiliza la palabra clave join. Las asociaciones se "desreferencian" utilizando la notación punto. Uniones implicit pueden aparecer en cualquiera de las cláusulas HQL. La unión implicit causa uniones internas (inner joins) en la declaración SQL que resulta.

```
from Cat as cat where cat.mate.name like '%s%'
```

## 15.5. Referencia a la propiedad identificadora

Hay dos maneras de referirse a la propiedad identificadora de una entidad:

- La propiedad especial (en minúsculas) id se puede utilizar para referenciar la propiedad identificadora de una entidad *dado que la entidad no defina un id del nombre de la propiedad no-identificadora*.
- Si la entidad define una propiedad identificadora nombrada, puede utilizar ese nombre de propiedad.

Las referencias a propiedades identificadoras compuestas siguen las mismas reglas de nombramiento. Si la entidad no tiene un id del nombre de la propiedad no-identificadora, la propiedad identificadora compuesta sólamente puede ser referenciada por su nombre definido. De otra manera se puede utilizar la propiedad id especial para referenciar la propiedad identificadora.

> **Importante**
>
> Observe que esto ha cambiado bastante desde la version 3.2.2. En versiones previas, id*siempre* se refería a la propiedad identificadora sin importar su nombre real. Una ramificación

de esa decisión fue que las propiedades no-identificadoras nombradas id nunca podrían ser referenciadas en consultas de Hibernate.

## 15.6. La cláusula select

La cláusula select escoge qué objetos y propiedades devolver en el conjunto de resultados de la consulta. Considere lo siguiente:

```
select mate
from Cat as cat
    inner join cat.mate as mate
```

La consulta seleccionará mates de otros Cats. Puede expresar esta consulta de una manera más compacta así:

```
select cat.mate from Cat cat
```

Las consultas pueden retornar propiedades de cualquier tipo de valor incluyendo propiedades del tipo componente:

```
select cat.name from DomesticCat cat
where cat.name like 'fri%'
```

```
select cust.name.firstName from Customer as cust
```

Las consultas pueden retornar múltiples objetos y/o propiedades como un array de tipo Object[],

```
select mother, offspr, mate.name
from DomesticCat as mother
    inner join mother.mate as mate
    left outer join mother.kittens as offspr
```

O como una List:

```
select new list(mother, offspr, mate.name)
from DomesticCat as mother
    inner join mother.mate as mate
    left outer join mother.kittens as offspr
```

O asumiendo que la clase Family tiene un constructor apropiado - como un objeto Java de tipo seguro:

```
select new Family(mother, mate, offspr)
from DomesticCat as mother
    join mother.mate as mate
    left join mother.kittens as offspr
```

Puede asignar alias para expresiones seleccionadas utilizando as:

```
select max(bodyWeight) as max, min(bodyWeight) as min, count(*) as n
from Cat cat
```

Esto es lo más útil cuando se usa junto con select new map:

```
select new map( max(bodyWeight) as max, min(bodyWeight) as min, count(*) as n )
from Cat cat
```

Esta consulta devuelve un Map de alias a valores seleccionados.

## 15.7. Funciones de agregación

Las consultas HQL pueden incluso retornar resultados de funciones de agregación sobre propiedades:

```
select avg(cat.weight), sum(cat.weight), max(cat.weight), count(cat)
from Cat cat
```

Las funciones de agregación soportadas son:

- avg(...), sum(...), min(...), max(...)
- count(*)
- count(...), count(distinct ...), count(all...)

Puede utilizar operadores aritméticos, concatenación y funciones SQL reconocidas en la cláusula select:

```
select cat.weight + sum(kitten.weight)
from Cat cat
    join cat.kittens kitten
group by cat.id, cat.weight
```

```
select firstName||' '||initial||' '||upper(lastName) from Person
```

Las palabras clave distinct y all se pueden utilizar y tienen las misma semántica que en SQL.

```
select distinct cat.name from Cat cat

select count(distinct cat.name), count(cat) from Cat cat
```

## 15.8. Consultas polimórficas

Una consulta como:

```
from Cat as cat
```

devuelve instancias no sólamente de Cat, sino también de subclases como DomesticCat. Las consultas de Hibernate pueden nombrar *cualquier* clase o interfaz Java en la cláusula from. La consulta retornará instancias de todas las clases persistentes que extiendan esa clase o implementen la interfaz. La siguiente consulta retornaría todos los objetos persistentes.

```
from java.lang.Object o
```

La interfaz Named se podría implementar por varias clases persistentes:

```
from Named n, Named m where n.name = m.name
```

Las dos últimas consultas requerirán más de un SELECT SQL. Esto significa que la cláusula order by no ordenará correctamente todo el conjunto que resulte. También significa que no puede llamar estas consulta usando Query.scroll().

## 15.9. La cláusula where

La cláusula where le permite refinar la lista de instancias retornadas. Si no existe ningún alias, puede referirse a las propiedades por nombre:

```
from Cat where name='Fritz'
```

Si existe un alias, use un nombre de propiedad calificado:

```
from Cat as cat where cat.name='Fritz'
```

Esto retorna instancias de Cat llamadas 'Fritz'.

La siguiente petición:

```
select foo
from Foo foo, Bar bar
where foo.startDate = bar.date
```

retornará todas las instancias de Foo con una instancia de bar con una propiedad date igual a la propiedad startDate del Foo. Las expresiones de ruta compuestas hacen la cláusula where extremadamente potente. Tome en consideración lo siguiente:

```
from Cat cat where cat.mate.name is not null
```

Esta consulta se traduce a una consulta SQL con una unión de tabla (interna). Por ejemplo:

```
from Foo foo
where foo.bar.baz.customer.address.city is not null
```

terminaría con una consulta que requeriría cuatro uniones de tablas en SQL.

El operador = se puede utilizar para comparar no sólamente propiedades sino también instancias:

```
from Cat cat, Cat rival where cat.mate = rival.mate
```

```
select cat, mate
from Cat cat, Cat mate
where cat.mate = mate
```

The special property (lowercase) id can be used to reference the unique identifier of an object. See Sección 15.5, “Referencia a la propiedad identificadora ” for more information.

```
from Cat as cat where cat.id = 123

from Cat as cat where cat.mate.id = 69
```

La segunda consulta es eficiente y no se necesita una unión de tablas.

También se pueden utilizar las propiedades de identificadores compuestos. Considere el siguiente ejemplo en donde Person tiene identificadores compuestos que consisten de country y medicareNumber:

```
from bank.Person person
where person.id.country = 'AU'
    and person.id.medicareNumber = 123456
```

```
from bank.Account account
where account.owner.id.country = 'AU'
    and account.owner.id.medicareNumber = 123456
```

Una vez más, la segunda consulta no requiere una unión de tablas.

See Sección 15.5, “Referencia a la propiedad identificadora ” for more information regarding referencing identifier properties)

La propiedad especial class acccede al valor discriminador de una instancia en el caso de persistencia polimórfica. Un nombre de clase Java incluído en la cláusula where será traducido a su valor discriminador.

```
from Cat cat where cat.class = DomesticCat
```

You can also use components or composite user types, or properties of said component types. See Sección 15.17, “Componentes” for more information.

Un tipo "any" tiene las propiedades especiales id y class, permiténdole expresar una unión de la siguiente forma (en donde AuditLog.item es una propiedad mapeada con <any>).

```
from AuditLog log, Payment payment
where log.item.class = 'Payment' and log.item.id = payment.id
```

La log.item.class y payment.class harían referencia a los valores de columnas de la base de datos completamente diferentes en la consulta anterior.

## 15.10. Expresiones

Las expresiones utilizadas en la cláusula where incluyen lo siguiente:

- operadores matemáticos: +, -, *, /
- operadores de comparación binarios: =, >=, <=, <>, !=, like
- operadores lógicos and, or, not
- Paréntesis ( ) que indican agrupación
- in, not in, between, is null, is not null, is empty, is not empty, member of y not member of
- Caso "simple", case ... when ... then ... else ... end, y caso "buscado", case when ... then ... else ... end
- concatenación de cadenas ...||... o concat(...,...)
- current_date(), current_time() y current_timestamp()
- second(...), minute(...), hour(...), day(...), month(...), and year(...)
- Cualquier función u operador definido por EJB-QL 3.0: substring(), trim(), lower(), upper(), length(), locate(), abs(), sqrt(), bit_length(), mod()
- coalesce() y nullif()
- str() para convertir valores numéricos o temporales a una cadena legible.
- cast(... as ...), donde el segundo argumento es el nombre de un tipo de Hibernate , y extract(... from ...) si cast() y extract() es soportado por la base de datos subyacente.
- la función index() de HQL, que se aplica a alias de una colección indexada unida.
- Las funciones de HQL que tomen expresiones de ruta valuadas en colecciones: size(), minelement(), maxelement(), minindex(), maxindex(), junto con las funciones especiales elements() e indices, las cuales se pueden cuantificar utilizando some, all, exists, any, in.
- Cualquier función escalar SQL soportada por la base de datos como sign(), trunc(), rtrim() y sin()
- parámetros posicionales JDBC ?
- parámetros con nombre :name, :start_date y :x1
- literales SQL 'foo', 69, 6.66E+2, '1970-01-01 10:00:01.0'
- constantes Java public static finaleg.Color.TABBY

in y between pueden utilizarse así:

```
from DomesticCat cat where cat.name between 'A' and 'B'
```

```
from DomesticCat cat where cat.name in ( 'Foo', 'Bar', 'Baz' )
```

Las formas negadas se pueden escribir así:

```
from DomesticCat cat where cat.name not between 'A' and 'B'
```

```
from DomesticCat cat where cat.name not in ( 'Foo', 'Bar', 'Baz' )
```

De manera similar, is null y is not null se pueden utilizar para probar valores nulos.

Los valores booleanos se pueden utilizar fácilmente en expresiones declarando substituciones de consulta HQL en la configuración de Hibernate:

```
<property name="hibernate.query.substitutions"
>true 1, false 0</property
>
```

Esto remplazará las palabras clave true y false con los literales 1 y 0 en el SQL traducido de este HQL:

```
from Cat cat where cat.alive = true
```

Puede comprobar el tamaño de una colección con la propiedad especial size o la función especial size().

```
from Cat cat where cat.kittens.size
> 0
```

```
from Cat cat where size(cat.kittens)
> 0
```

Para las colecciones indexadas, puede referirse a los índices máximo y mínimo utilizando las funciones minindex y maxindex. De manera similar, se puede referir a los elementos máximo y mínimo de una colección de tipo básico utilizando las funciones minelement y maxelement. Por ejemplo:

```
from Calendar cal where maxelement(cal.holidays)
> current_date
```

```
from Order order where maxindex(order.items)
> 100
```

```
from Order order where minelement(order.items)
> 10000
```

Las funciones SQL any, some, all, exists, in están soportadas cuando se les pasa el conjunto de elementos o índices de una colección (las funciones elements e indices) o el resultado de una subconsulta (vea a continuación):

```
select mother from Cat as mother, Cat as kit
where kit in elements(foo.kittens)
```

```
select p from NameList list, Person p
where p.name = some elements(list.names)
```

```
from Cat cat where exists elements(cat.kittens)
```

```
from Player p where 3
> all elements(p.scores)
```

```
from Show show where 'fizard' in indices(show.acts)
```

Note que estas construcciones - size, elements, indices, minindex, maxindex, minelement, maxelement - solo se pueden utilizar en la cláusula where en Hibernate3.

Los elementos de colecciones indexadas (arrays, listas, mapas) se pueden referir por índice sólamente en una cláusula where:

```
from Order order where order.items[0].id = 1234
```

```
select person from Person person, Calendar calendar
where calendar.holidays['national day'] = person.birthDay
    and person.nationality.calendar = calendar
```

```
select item from Item item, Order order
where order.items[ order.deliveredItemIndices[0] ] = item and order.id = 11
```

```
select item from Item item, Order order
where order.items[ maxindex(order.items) ] = item and order.id = 11
```

La expresión dentro de [] puede incluso ser una expresión aritmética:

```
select item from Item item, Order order
where order.items[ size(order.items) - 1 ] = item
```

HQL también proporciona la función incorporada index(), para los elementos de una asociación uno-a-muchos o una colección de valores.

```
select item, index(item) from Order order
    join order.items item
where index(item) < 5
```

Se pueden utilizar las funciones SQL escalares soportadas por la base de datos subyacente:

```
from DomesticCat cat where upper(cat.name) like 'FRI%'
```

Considere qué tan larga y menos leíble sería la siguiente consulta en SQL:

```
select cust
from Product prod,
    Store store
    inner join store.customers cust
where prod.name = 'widget'
    and store.location.name in ( 'Melbourne', 'Sydney' )
    and prod = all elements(cust.currentOrder.lineItems)
```

*Ayuda:* algo como

```
SELECT cust.name, cust.address, cust.phone, cust.id, cust.current_order
FROM customers cust,
    stores store,
    locations loc,
    store_customers sc,
    product prod
WHERE prod.name = 'widget'
    AND store.loc_id = loc.id
    AND loc.name IN ( 'Melbourne', 'Sydney' )
    AND sc.store_id = store.id
    AND sc.cust_id = cust.id
    AND prod.id = ALL(
        SELECT item.prod_id
        FROM line_items item, orders o
        WHERE item.order_id = o.id
            AND cust.current_order = o.id
    )
```

## 15.11. La cláusula order by

La lista retornada por una consulta se puede ordenar por cualquier propiedad de una clase retornada o componentes:

```
from DomesticCat cat
order by cat.name asc, cat.weight desc, cat.birthdate
```

Los asc o desc opcionales indican ordenamiento ascendente o descendente respectivamente.

## 15.12. La cláusula group by

Una consulta que retorna valores agregados se puede agrupar por cualquier propiedad de una clase retornada o componentes:

```
select cat.color, sum(cat.weight), count(cat)
from Cat cat
group by cat.color
```

```
select foo.id, avg(name), max(name)
from Foo foo join foo.names name
group by foo.id
```

Se permite también una cláusula having.

```
select cat.color, sum(cat.weight), count(cat)
from Cat cat
group by cat.color
having cat.color in (eg.Color.TABBY, eg.Color.BLACK)
```

Las funciones SQL y las funciones de agregación SQL están permitidas en las cláusulas having y order by, si están soportadas por la base de datos subyacente (por ejemplo, no lo están en MySQL).

```
select cat
from Cat cat
    join cat.kittens kitten
group by cat.id, cat.name, cat.other, cat.properties
having avg(kitten.weight)
> 100
order by count(kitten) asc, sum(kitten.weight) desc
```

La cláusula group by ni la cláusula order by pueden contener expresiones aritméticas. Hibernate tampocoo expande una entidad agrupada así que no puede escribir group by cat si todas las propiedades de cat son no-agregadas. Tiene que enumerar todas la propiedades no-agregadas explícitamente.

## 15.13. Subconsultas

Para bases de datos que soportan subconsultas, Hibernate soporta subconsultas dentro de consultas. Una subconsulta se debe encerrar entre paréntesis (frecuentemente por una llamada a una función de agregación SQL). Incluso se permiten subconsultas correlacionadas (subconsultas que se refieren a un alias en la consulta exterior).

```
from Cat as fatcat
where fatcat.weight
> (
    select avg(cat.weight) from DomesticCat cat
)
```

```
from DomesticCat as cat
where cat.name = some (
    select name.nickName from Name as name
)
```

```
from Cat as cat
where not exists (
    from Cat as mate where mate.mate = cat
)
```

```
from DomesticCat as cat
where cat.name not in (
    select name.nickName from Name as name
)
```

```
select cat.id, (select max(kit.weight) from cat.kitten kit)
from Cat as cat
```

Note que las subconsultas HQL pueden ocurrir sólamente en las cláusulas select o where.

Note that subqueries can also utilize row value constructor syntax. See Sección 15.18, "Sintaxis del constructor de valores por fila" for more information.

## 15.14. Ejemplos de HQL

Las consultas de Hibernate pueden ser bastante potentes y complejas. De hecho, el poder del lenguaje de consulta es uno de las fortalezas principales de Hibernate. He aquí algunos ejemplos de consultas muy similares a las consultas de proyectos recientes. Note que la mayoría de las consultas que escribirá son mucho más simples que los siguientes ejemplos.

La siguiente consulta retorna el order id, número de items y valor total mínimo dado y el valor de la orden para todas las órdenes no pagadas de un cliente en particular. Los resultados se ordenan de acuerdo al valor total. Al determinar los precios, usa el catálogo actual. La consulta SQL resultante, contra las tablas ORDER, ORDER_LINE, PRODUCT, CATALOG y PRICE tiene cuatro uniones interiores y una subselección (no correlacionada).

```
select order.id, sum(price.amount), count(item)
from Order as order
    join order.lineItems as item
    join item.product as product,
    Catalog as catalog
    join catalog.prices as price
where order.paid = false
    and order.customer = :customer
    and price.product = product
    and catalog.effectiveDate < sysdate
    and catalog.effectiveDate
>= all (
        select cat.effectiveDate
        from Catalog as cat
        where cat.effectiveDate < sysdate
    )
group by order
having sum(price.amount)
> :minAmount
order by sum(price.amount) desc
```

¡Qué monstruo! Realmente, en la vida real, no me gustan mucho las subconsultas, de modo que mi consulta fue realmente algo como esto:

```
select order.id, sum(price.amount), count(item)
from Order as order
    join order.lineItems as item
    join item.product as product,
    Catalog as catalog
    join catalog.prices as price
where order.paid = false
    and order.customer = :customer
    and price.product = product
    and catalog = :currentCatalog
group by order
having sum(price.amount)
> :minAmount
order by sum(price.amount) desc
```

La próxima consulta cuenta el número de pagos en cada estado, excluyendo todos los pagos en el estado AWAITING_APPROVAL donde el cambio más reciente al estado lo hizo el usuario actual. Se traduce en una consulta SQL con dos uniones interiores y una subselección correlacionada contra las tablas PAYMENT, PAYMENT_STATUS y PAYMENT_STATUS_CHANGE.

```
select count(payment), status.name
from Payment as payment
    join payment.currentStatus as status
    join payment.statusChanges as statusChange
where payment.status.name <
> PaymentStatus.AWAITING_APPROVAL
    or (
```

```
            statusChange.timeStamp = (
                select max(change.timeStamp)
                from PaymentStatusChange change
                where change.payment = payment
            )
            and statusChange.user <
> :currentUser
        )
group by status.name, status.sortOrder
order by status.sortOrder
```

Si la colección statusChanges se mapeara como una lista, en vez de un conjunto, la consulta habría sido mucho más simple de escribir.

```
select count(payment), status.name
from Payment as payment
    join payment.currentStatus as status
where payment.status.name <
> PaymentStatus.AWAITING_APPROVAL
    or payment.statusChanges[ maxIndex(payment.statusChanges) ].user <
> :currentUser
group by status.name, status.sortOrder
order by status.sortOrder
```

La próxima consulta utiliza la función isNull() de MS SQL Server para devolver todas las cuentas y pagos aún no cancelados de la organización a la que pertenece el usuario actual. Se traduce como una consulta SQL con tres uniones interiores, una unión exterior y una subselección contra las tablas ACCOUNT, PAYMENT, PAYMENT_STATUS, ACCOUNT_TYPE, ORGANIZATION y ORG_USER.

```
select account, payment
from Account as account
    left outer join account.payments as payment
where :currentUser in elements(account.holder.users)
    and PaymentStatus.UNPAID = isNull(payment.currentStatus.name, PaymentStatus.UNPAID)
order by account.type.sortOrder, account.accountNumber, payment.dueDate
```

Para algunas bases de datos, necesitaríamos eliminar la subselección (correlacionada).

```
select account, payment
from Account as account
    join account.holder.users as user
    left outer join account.payments as payment
where :currentUser = user
    and PaymentStatus.UNPAID = isNull(payment.currentStatus.name, PaymentStatus.UNPAID)
order by account.type.sortOrder, account.accountNumber, payment.dueDate
```

## 15.15. Declaraciones UPDATE y DELETE masivas

HQL now supports update, delete and insert ... select ... statements. See Sección 14.4, “Operaciones de estilo DML” for more information.

## 15.16. Consejos y Trucos

Puede contar el número de resultados de una consulta sin retornarlos:

```
( (Integer) session.createQuery("select count(*) from ....").iterate().next() ).intValue()
```

Para ordenar un resultado por el tamaño de una colección, utilice la siguiente consulta:

```
select usr.id, usr.name
from User as usr
    left join usr.messages as msg
group by usr.id, usr.name
order by count(msg)
```

Si su base de datos soporta subselecciones, puede colocar una condición sobre el tamaño de selección en la cláusula where de su consulta:

```
from User usr where size(usr.messages)
>= 1
```

Si su base de datos no soporta subselecciones, utilice la siguiente consulta:

```
select usr.id, usr.name
from User usr
    join usr.messages msg
group by usr.id, usr.name
having count(msg)
>= 1
```

Como esta solución no puede retornar un User con cero mensajes debido a la unión interior, la siguiente forma también es útil:

```
select usr.id, usr.name
from User as usr
    left join usr.messages as msg
group by usr.id, usr.name
having count(msg) = 0
```

Las propiedades de un JavaBean pueden ser ligadas a los parámetros de consulta con nombre:

```
Query q = s.createQuery("from foo Foo as foo where foo.name=:name and foo.size=:size");
q.setProperties(fooBean); // fooBean has getName() and getSize()
List foos = q.list();
```

Las colecciones son paginables usando la interfaz Query con un filtro:

```
Query q = s.createFilter( collection, "" ); // the trivial filter
q.setMaxResults(PAGE_SIZE);
q.setFirstResult(PAGE_SIZE * pageNumber);
List page = q.list();
```

Los elementos de colección se pueden ordenar o agrupar usando un filtro de consulta:

```
Collection orderedCollection = s.filter( collection, "order by this.amount" );
Collection counts = s.filter( collection, "select this.type, count(this) group by this.type" );
```

Puede hallar el tamaño de una colección sin inicializarla:

```
( (Integer) session.createQuery("select count(*) from ....").iterate().next() ).intValue();
```

## 15.17. Componentes

Los componentes se pueden utilizar de la misma manera en que se pueden utilizar los tipos de valores simples en consultas HQL. Pueden aparecer en la cláusula select así:

```
select p.name from Person p
```

```
select p.name.first from Person p
```

en donde el nombre de la Persona es un componente. Los componentes también se pueden utilizar en la cláusula where:

```
from Person p where p.name = :name
```

```
from Person p where p.name.first = :firstName
```

Los componentes también se pueden utilizar en la cláusula where:

```
from Person p order by p.name
```

```
from Person p order by p.name.first
```

Otro uso común de los componentes se encuentra en row value constructors.

## 15.18. Sintaxis del constructor de valores por fila

HQL soporta la utilización de la sintaxis row value constructor de SQL ANSI que a veces se denomina sintaxis tuple, aunque puede que la base de datos subyacentes no soporte esa noción. Aquí estamos refiriéndonos generalmente a las comparaciones multivaluadas que se asocian típicamente con los componentes. Considere una entidad Persona, la cual define un componente de nombre:

```
from Person p where p.name.first='John' and p.name.last='Jingleheimer-Schmidt'
```

Esa es una sintaxis válida aunque un poco verbosa. Puede hacerlo un poco más conciso utilizando la sintaxis row value constructor:

```
from Person p where p.name=('John', 'Jingleheimer-Schmidt')
```

También puede ser útil especificar esto en la cláusula select:

```
select p.name from Person p
```

También puede ser beneficioso el utilizar la sintaxis row value constructor cuando se utilizan subconsultas que necesitan compararse con valores múltiples:

```
from Cat as cat
where not ( cat.name, cat.color ) in (
    select cat.name, cat.color from DomesticCat cat
)
```

Algo que se debe tomar en consideración al decidir si quiere usar esta sintaxis es que la consulta dependerá del orden de las sub-propiedades componentes en los metadatos.