Apuntes accesibles DQL, (selects unitabla)

Sitio: FRANCISCO DE GOYA
Curso: Bases de datos ASIR1, DAM1 y DAW1
Libro: Apuntes accesibles DQL, (selects unitabla)

Imprimido por: Alberto Bollo Rodríguez
Día: jueves, 11 de enero de 2024, 13:21

Tabla de contenidos

1. Introducción

El lenguaje SQL comprende dentro de un SGBD la funcionalidad de creación de estructuras (DDL), la modificación de los datos (DML), así como la consulta (DQL). Y probablemente es esta parte de consultas la más utilizada.

El lenguaje SQL fue estandarizado por ANSI en 1986, y posteriormente, en 1992, revisado como SQL2 o SQL-92. El estándar más reciente es el SQL3 (creado en 2003), que entre otras novedades, incorpora funcionalidades relacionadas con XML, el tipo de datos autonumérico, expresiones regulares, consultas recursivas, disparadores, tipos de datos no escalares, y aspectos de orientación a objetos.

Muchos sistemas comerciales no se han adaptado aún al SQL3. De hecho, incluso, es difícil encontrar sistemas que implementen todas las funcionalidades previstas en el SQL-92 o que no presenten ciertas desviaciones del estándar. Las sentencias que se describen a continuación son las más comúnmente utilizadas de SQL92 y, en algunos casos, variantes específicas de MySQL.

2. Consultas unitabla

Con Unitabla nos referimos a consultas que obtienen su información de una sola tabla como origen o fuente de los datos.

La operación de recuperación de datos (o consulta) por excelencia en las bases de datos relacionales es la Selección. Con este nombre se da la idea de que escogemos recuperar ciertos datos, basándonos en un criterio de selección o búsqueda. Por ejemplo, una consulta de selección puede ser aquella en la que se recuperan todos los empleados (tuplas completas, es decir, con todos los campos) con un salario mayor a los 3000€ pero inferior a 6000€.

Sobre la selección también se puede realizar una Proyección de cada tupla o registro. Es decir podemos optar por no recuperar todos los campos o atributos sino un subconjunto. Por ejemplo, en la consulta antes mencionada podríamos obtener sólo los atributos nombre, apellidos y sueldo de los Empleados con un salario mayor a los 3000€ pero inferior a 6000€.

Se realizan mediante el comando SELECT, cuya sintaxis (en notación BNF) es:

```
SELECT [ DISTINCT | ALL ] { * | <campo> [as <alias> [,<campo>] [as <alias>] }

FROM (<lista_tablas> {<alias} | <tabla_join> ) {,<lista_tablas> {<alias} | <tabla_join>

[ON <condicion>]}

WHERE <condición>;

GROUP BY <atributo> {, <atributo>}

[ HAVING <condición_de_selección_de_grupo> ]

ORDER BY <atributo> [ASC|DESC] {, <atributo> [orden]}

LIMIT <num>;
```

Donde:

```
<lista_atributos> ::= (*|(<atributo>|<funcion>(([DISTINCT] <atributo>)|*)))
```

La consulta mínima ha de contener un select y valores fijos (posiblemente con funciones), aunque es de poca utilidad más allá de la práctica de funciones

```
select 2+2;
```

Pero por lo general la consulta mínima indicará una serie de campos o * (todos los campos) y una tabla de donde provienen:

```
select nombre, apellido from persona;
```

2.1. Consultas sencillas

Vamos a verlo con ejemplos utilizando la base de datos MASCOTAS, en mysql, que contiene la siguiente tabla:

```
SELECT * FROM Mascotas;
```

Mediante esta consulta se va a seleccionar todas las columnas de la tabla, debido a que el * siempre se refiere a todos los campos, y al no haber cláusula ***where*** también todas las filas.

nombre	propietario	especie	sexo	nacimiento	fallecimiento
Fluffy	David	Gato	f	2009-02-04	NULL
Mau	Juan	Gato	m	2008-03-17	NULL
Buffy	David	Perro	f	2009-05-13	NULL
FanFan	Benito	Perro	m	2010-08-27	NULL
Kaiser	Diana	Perro	m	2014-05-23	2017-07-29
Chispa	Omar	Ave	f	2008-09-11	NULL
Wicho	Tomás	Ave	NULL	2010-02-09	NULL
Skim	Benito	Serpiente	m	2011-04-29	NULL

Es posible por lo tanto no seleccionar todas las columnas, si se indica cuáles queremos visualizar:

```
select nombre, nacimiento from mascotas;
```

nombre	nacimiento
Fluffy	2009-02-04
Mau	2008-03-17
Buffy	2009-05-13
FanFan	2010-08-27
Kaiser	2014-05-23
Chispa	2008-09-11
Wicho	2010-02-09
Skim	2011-04-29

Como también es posible repetirlas

```
select nombre, nacimiento, nombre from mascotas;
```

nombre	nacimiento	nombre
Fluffy	2009-02-04	Fluffy
Mau	2008-03-17	Mau
Buffy	2009-05-13	Buffy
FanFan	2010-08-27	FanFan
Kaiser	2014-05-23	Kaiser
Chispa	2008-09-11	Chispa
Wicho	2010-02-09	Wicho
Skim	2011-04-29	Skim

2.2. Cláusula DISTINCT

La cláusula distinct va a evitar las repeticiones de un solo campo:

```
SELECT DISTINCT propietario FROM Mascotas;
```

propietario
David
Juan
Benito
Diana
Omar
Tomás

En caso de incluir más de un campo, lo que no se repetirá es la combinación de ambos:

```
select distinct sexo, especie from mascotas;
```

sexo	especie
f	Gato
m	Gato
f	Perro
m	Perro
f	Ave
NULL	Ave
m	Serpiente

Si se observa en la tabla original, hay dos perros con sexo m, pero en la consulta anterior solo aparece uno

2.3. Alias

Es posible renombrar algunos de los atributos recuperados o los registros resultantes con un nombre de tabla. Esto se conoce también como uso de alias en las sentencias SQL.

```
SELECT nombre AS apodo, especie FROM Mascotas;
```

apodo	especie
Fluffy	Gato
Mau	Gato
Buffy	Perro
FanFan	Perro
Kaiser	Perro
Chispa	Ave
Wicho	Ave
Skim	Serpiente

Esto será realmente útil cuando se usen mútiples funciones sobre un campo, ya que al ser tan extenso, cuando se muestra la tabla el ancho de la columna es excesivo. Pero sobre todo cuando más adelante se realicen subconsultas dado que se ha de referir mediante un nombre a un campo de una consulta a otra.

2.4. Cláusula ORDER BY

Podemos ordenar las tuplas resultantes por uno o varios campos, en orden ascendente o descentente, mediante la cláusula ORDER BY:

```
SELECT nombre, nacimiento FROM Mascotas ORDER BY nacimiento;
```

nombre	nacimiento
Mau	2008-03-17
Chispa	2008-09-11
Fluffy	2009-02-04
Buffy	2009-05-13
Wicho	2010-02-09
FanFan	2010-08-27
Skim	2011-04-29
Kaiser	2014-05-23

Se puede invertir el orden mediante la cláusula DESC después del criterio de orden:

```
SELECT nombre, nacimiento FROM Mascotas ORDER BY nacimiento DESC;
```

nombre	nacimiento
Kaiser	2014-05-23
Skim	2011-04-29
FanFan	2010-08-27
Wicho	2010-02-09
Buffy	2009-05-13
Fluffy	2009-02-04
Chispa	2008-09-11
Mau	2008-03-17

Además, es posible ordenar por más de un campo. El orden resultante es el resultado de ordenar por el primer campo que se indique, y solo se aplica el segundo (o posteriores campos) en caso de que el primero sea igual en varios casos.

```
SELECT nombre, especie, nacimiento FROM Mascotas ORDER BY especie, nacimiento DESC;
```

nombre	especie	nacimiento
Wicho	Ave	2010-02-09
Chispa	Ave	2008-09-11
Fluffy	Gato	2009-02-04
Mau	Gato	2008-03-17
Kaiser	Perro	2014-05-23
FanFan	Perro	2010-08-27
Buffy	Perro	2009-05-13
Skim	Serpiente	2011-04-29

3. Consultas con filtro

En la cláusula where de la sentencia select se permite establecer criterios que deberán satisfacer las filas de la tabla seleccionada para aparecer en el resultado.

Los criterios pueden ser una o varias condiciones. Cada criterio deberá tener un resultado "cierto" o "falso":

- sexo = "f"
- especie = "perro"
- nombre = "Kaiser"

O algo más complejas:

- length(nombre) > 5
- year(fechaNac) < 2005

De haber varias condiciones deben unirse mediante operadores lógicos and, or y not:

- sexo = "f" and !(especie = "perro")
- especie = "perro" or especie = "gato"

También pueden usarse los operadores usuales en otros lenguajes de programación:

- sexo = "f" || !(especie = "Perro")

El último ejemplo es todas las mascotas hembra o que no sean perros:

```
select nombre, especie, sexo from mascotas where sexo = "f" or !(especie = "Perro");
```

nombre	especie	sexo
Fluffy	Gato	f
Mau	Gato	m
Buffy	Perro	f
Chispa	Ave	f
Wicho	Ave	null
Skim	Serpiente	m

Otra consulta sería:

```
select nombre, especie, nacimiento from mascotas where especie = "Perro" or especie = "Gato";
```

nombre	especie	nacimiento
Fluffy	Gato	1999-02-04
Mau	Gato	1998-03-17
Buffy	Perro	1999-05-13
FanFan	Perro	2000-08-27
Kaiser	Perro	2004-05-23

3.1. Operadores de condición

En cualquiera de las sentencias que se han mostrado anteriormente se pueden utilizar condiciones. Estas condiciones sirven para que la acción afecte únicamente a las filas que cumplan las condiciones, por ejemplo a la hora de borrar filas o de actualizarlas.

Los comparadores son similares a cualquier lenguaje de programación:

- Igual: = (un solo igual, no dos)
- No igual: !=
- Menor, mayor, etc: <, >, <=, >=
- SImilar: like
- Comparación con null: is null, is not null

La condición ha de tener un comparador entre dos elementos, que habitualmente son un campo de una tabla (sobre la que se hace un update o delete) y un valor estático, aunque puede darse cualquier combinación, incluso entre campos de la misma o distintas tablas. También se puede comparar un campo con el resultado de un select.

De esta forma un ejemplo sería:

```
delete from tabla where edad > 22;
```

Eliminará todas las filas cuyo campo edad sea superior a 22. En caso de querer borrar un único elemento, se deberá comparar con el identificador:

```
delete from persona where dni = "12345678A";
```

3.2. Operador like

El comparador like es un comparador específico de mysql, aunque su funcionalidad se puede conseguir en otros lenguajes con funciones.

Lo que hace es comparar entre dos cadenas si estas son similares. Para ello se sirve de dos metacaracteres.

%: se sustituye por 0, 1 o cualquier número de caracteres cualesquiera.

_: se sustituye obligatoriamente por un carácter cualquiera.

Así tendríamos:

nombre like "a%" : nombre empieza por a.

nombre like "a_" : nombre de dos caracteres siendo el primero una a.

nombre like "%a" : nombre acaba por a.

nombre like "%a%": nombre contiene al menos una a.

Las combinaciones son enormes por lo que es una herramienta muy potente.

3.3. Operadores y prioridad

Los operadores se pueden aplicar a cualquier tipo de datos, aunque puedan dar resultados no esperados. Con números, operan como se espera.

Con fechas hay que saber que internamente se almacenan como números, de forma que "<" y ">" seguirá sirviendo para antes y después, pero sumar o restar no tiene el sentido que se espera, para ello se usa timestampdiff(), date_add() y otras funciones que se verán en su propia sección.

Con las cadenas de caracteres, los operadores + y - devolverán un 0, no tienen sentido, sin embargo ">" y "<" servirá para comparar alfabéticamente.

Los operadores tienen una prioridad. Al igual que en matemáticas 4+3*2 el orden es multiplicar primero 3*2 y posteriormente sumar 4, todos los operadores en SQL tienen una prioridad.

- Igual: =
- Igual teniendo en cuenta null: <=>
- Diferente: <>, !=
- Mayor, menor, mayor igual, menor igual: >, <, >=, <=
- Valores en una lista: IN
- Valores entre: between
- Nulos: is null, is not null
- Patrón: like
- Si una consulta devuelve al menos una fila: exists

En MySql no existe el tipo booleano (true-false), sino que se usa un int, donde el 0 es falso (false) y el 1 es verdadero (true).

operación	resultado
7+2*3	13
(7-2)*3	15
7>2	1
9<2	0
7>2 and 4>3	1
7>2 or 4<3	1
(10 >= 10 and 0 <= 1) +2	3

Los valores nulos se escriben sin comillas: null. Dado que "null" es un varchar con cuatro caracteres y por lo tanto no es nulo.

El orden de los operadores será:

- Paréntesis ( )
- Multiplicación, división y resto (módulo).
- Suma, resta, concatenación
- Comparaciones: <, >, =, !=...
- Negación not, !
- and (&& está en desuso y en futuras versiones no funcionará)
- or (|| está en desuso y en futuras versiones no funcionará)

Ejemplos:

```
select nombre, nacimiento from mascotas where year(nacimiento) between 2000 and 2001;
```

nombre	nacimiento
FanFan	2000-08-27
Wicho	2000-02-09
Skim	2001-04-29

```
select nombre, nacimiento from mascotas where propietario in ("David","Diana","Juan");
```

nombre	nacimiento
Fluffy	1999-02-04
Mau	1998-03-17
Buffy	1999-05-13
Kaiser	2004-05-23

3.4. Operadores con múltiples valores

Los operadores IN, ANY / SOME y ALL son útiles para comparar con varios valores, pero resultarán de mucha mayor utilidad cuando se aborden las subconsultas. En este punto, antes de subconsultas solo abordaremos IN.

Operador IN

El operador IN sirve para comparar un campo o valor con una serie de valores. Será cierto si el valor se encuentra el algunos de los valores que se ofrece a continuación del IN.

Por ejemplo, en la base de datos bebés, realizamos la siguiente consulta:

```
select * from nacimientos where provincia in ("H","S");
```

Obtendremos el siguiente resultado:

Nombre	Apellido1	Apellido2	FechaNac	Provincia
Alberto	Ferro	L¢pez	2005-01-20	H
Alberto	R¡os	Roma	2005-10-16	H
Felipe	Fernández	Gómez	2007-11-13	S

Realmente sería equivalente a que la condición se hiciese con = y OR:

```
select * from nacimientos where provincia = "H" or provincia = "S";
```

Pero si la lista de valores es larga será más cómodo. Y aún será más eficiente cuando la lista dependa de una subconsulta, como se verá más adelante.

4. Consultas de agregación

Las consultas de agregación utilizan ciertas funciones que resumen los datos existentes, en vez de filtrarlos. Pierden información por el camino, lo cual es importante, porque no se podrá recuperar información particular de una fila concreta, pero sí se puede obtener un mayor análisis de los datos, por ejemplo:

- Cuántos animales hay:

```
select count(*) from mascotas
```

count(*)
8

Esta es la versión más sencilla. Contar filas. La pérdida de información está en que no hay ningún dato que provenga de alguna de las filas de la tabla, pero lo que se gana es que se sabe cuántas filas hay, que por otra parte no es un dato que esté en ninguna fila.

Se puede hacer con muchas otras funciones, como por ejemplo max() sobre la tabla empleados de la BD miscelanea

```
select max(sueldo) from empleados;
```

max(sueldo)
1800

Como se ve en el ejemplo anterior, también es posible obtener algunas filas concretas que mediante filtros no sería posible. Por ejemplo, en mascotas se puede saber la edad en años de una mascota mediante la función timestampdiff(year,nacimiento,curdate()). Si se usa la función max() con esta, devolverá la edad máxima.

```
select max(timestampdiff(year,nacimiento,curdate())) as edad from mascotas;
```

edad
21

Pero se habrá perdido información, ya que para conocer quién es aquel que tiene la mayor edad se necesitaría una subconsulta

4.1. Agrupación

Cuando se usa una función de agrupación se obtiene una información relativa a todas las filas. Pero también es posible obtener la misma información pero agrupada por algún criterio.

- Seleccionar cuántos bebés han nacido por año:

```
select count(*), year(fechaNac) as anio from nacimientos group by anio;
```

count(*)	anio
3	2005
3	2006
8	2007
4	2008
6	2009

En este caso se ha usado la función contar de la misma forma, pero se le ha añadido la cláusula group by. Esta cláusula lo que permite es indicar una serie de campos y se realizará la cuenta de filas por cada grupo de valores que coincidan.

También se ve la utilidad de usar un alias. En vez de usar group by por toda la función year() y su contenido, simplemente se agrupa por el alias. Esto es aún más útil si se utiliza la selección de la página anterior sobre la base de datos de mascotas:

```
select count(*), timestampdiff(year,nacimiento,curdate()) as edad from mascotas group by edad;
```

count(*)	edad
1	15
1	18
2	19
2	20
2	21

Por supuesto se puede hacer la agrupación por varios campos, y la cuenta se realizará de forma que coincidan todos los campos de la lista group by, por ejemplo de nuevo en mascotas:

```
select count(*), especie, sexo from mascotas group by especie, sexo order by especie;
```

count(*)	especie	sexo
1	Ave	NULL
1	Ave	f
1	Gato	f
1	Gato	m
1	Perro	f
2	Perro	m
1	Serpiente	m

En este caso nos indica que hay un ejemplar de cada especie y sexo, excepto de la tupla "perro","m" que tiene dos.

4.2. Filtros y having

Hay que diferenciar dos condiciones que se pueden indicar cuando se realizan consultas de agregación.

Será posible filtrar las filas antes de que se aplique la función de agregación, esto es, que no se tendrán en cuenta a la hora de contar, hacer medias o hallar máximos entre otras. Estas condiciones, como es habitual, se indican en la cláusula WHERE.

Pero también se puede hacer un segundo filtro: mostrar únicamente los datos ya agrupados que cumplan unas condiciones. O lo que es lo mismo, hacer un filtro después de la agrupación, mostrando las filas ya agrupadas que cumplan condiciones. Esto se hace en la cláusula HAVING.

Por ejemplo, ver el número de nacimientos de bebés por provincia.

```
select count(*),provincia from nacimientos group by provincia;
```

count(*)	provincia
3	AL
1	B
1	H
18	M
1	TO

Pero se podría hacer lo mismo solo para los nacidos en 2007

```
select count(*),provincia from nacimientos where year(fechaNac) = 2007 group by provincia;
```

count(*)	provincia
1	B
7	M

Siendo así, se muestra cuántos nacidos hay por provincia, pero solo teniendo en cuenta los nacidos en 2007.

Mientras que si se quiere filtrar los resultados después de agrupar, sería de la siguiente forma

```
select count(*),provincia from nacimientos group by provincia having count(*)>1;
```

count(*)	provincia
3	AL
18	M

Esto sería mostrar los nacidos por provincia, pero solo de aquellos casos en los que haya nacido más de uno.

Aunque también es posible hacerlo con ambas, el filtro antes de agrupar (WHERE) y después de agrupar (HAVING). Por ejemplo en la tabla facturas.

```
select count(*)as num,left(nombre,1) as inicial from nacimientos where year(fechaNac) = 2007 group by inicial having num > 1;
```

num	inicial
2	F
2	R

Con esto se ha seleccionado el número de nacidos en 2007 por inicial del nombre, siempre que sean más de un nacimiento. Con esto se ignora los nacidos en otros años que no sean 2007, y no se muestran las iniciales con un solo nacimiento. Se puede probar que si se quita la cláusula HAVING, hay otro resultado:

```
select count(*)as num,left(nombre,1) as inicial from nacimientos where year(fechaNac) = 2007 group by inicial;
```

num	inicial
1	A
1	B
1	C
2	F
1	I
2	R

5. Otros elementos

Otros elementos se pueden tener en las consultas que dan más herramientas para realizar las selecciones necesarias.

5.1. Limit

La cláusula limit se usa para limitar el número de filas que se muestran. Se coloca siempre al final de la consulta.

Por ejemplo, la tabla Country de la base de datos World, tiene 239 filas. Sin filtro siempre aparecen todas. Así, si usamos limit:

```
select Name, Population from Country limit 4;
```

Name	Pupulation
Afghanistan	22720000
Netherlands	15864000
Netherlands Antilles	217000
Albania	3401200

Pero adquiere una mayor utilidad si se usa junto la cláusula order. Se podrá obtener las filas con mayor valor en una columna ordenando descendentemente, mientras que obtendremos las filas con menor valor si ordenamos de forma ascendente. En el siguiente ejemplo se muestran los 4 países más poblados.

```
select Name, Population from Country order by Population desc limit 4;
```

Name	Pupulation
China	1277558000
India	1013662000
United States	278357000
Indonesia	212107000

6. Errores típicos

Se detalla aquí una serie de errores típicos, que con los años más se suele reincidir en los mismos. Tanto fallos al hacer ejercicios como por alguna diferencia de configuración en la instalación del servidor MySQL en Windows.

6.1. only_full_group_by

Normalmente al instalar en cualquier Linux, por defecto está el modo only_full_group_by. Esto impide que se realicen consultas incompletas, en las que la respuesta puede tomar algún valor al azar (o arbitrariamente). Realmente debería dar un error.

La siguiente consulta debería dar error:

```
use miscelanea;

select count(*), cliente from facturas;
```

El error que debe dar es el siguiente:

```
ERROR 1140 (42000): In aggregated query without GROUP BY, expression #2 of SELECT list contains nonaggregated column 'miscelanea.facturas.cliente'; this is incompatible with sql_mode=only_full_group_by
```

En caso de que, en vez de dar error, devuelva algo así:

count(*)	cliente
114	Ahorra muchísimo

Significa que este modo only_full_group_by no está activo.

Se puede activar cada vez que se inicia el cliente mysql de la siguiente forma:

```
set sql_mode=only_full_group_by
```

Pero también puede hacerse permanente cambiando el fichero de configuración del servidor. Se puede encontrar en:

- linux: /etc/mysql/mysql.conf.d/mysqld.cnf
- Windows con MySQL solo: \Program Files\Mysql\Mysql Server 8.0\bin\my.ini
- Windows con xamp: \xamp\mysql\bin\my.ini

Buscar la opción sql_mode. Si no existe, crearla en la sección [mysqld] de la forma siguiente:

sql_mode=NO_ZERO_IN_DATE,NO_ZERO_DATE,NO_ENGINE_SUBSTITUTION,ONLY_FULL_GROUP_BY

En caso de que ya existiese y no contuviese only_full_group_by, añadirlo al final de las opciones sql_mode.

6.2. Uso de distinct en vez de group by

Cuando un ejercicio pide un resultado en el que, aparentemente, no se repiten ciertos valores de una columna, como por ejemplo, si queremos saber el importe de la suma del precio por unidad de todos los productos de cada cliente, de miscelánea:

```
select cliente, sum(precio_unidad) from facturas group by cliente;
```

Obtenemos la siguiente tabla:

cliente	sum(precio_unidad)
Compraqui	1.70
Super Paquito	65.42
Ahorra Muchisimo	32.97
Baratolandia	51.18
El Barato	17.99
El Campo	25.08
Blanca Lecheria	8.19

Podríamos tener la tentación de pensar que la columna izquierda se obtiene mediante un distinct. Esto no es correcto.

Con un ejemplo más sencillo se puede ver:

x	y
A	1
B	3
A	2
A	1
C	2
C	2
B	2

Si buscamos sumar por cada tipo de X (que aparezca un solo X) con una consulta similar a la siguiente:

```
select distinct x,y from T;
```

Obtendremos:

x	y
A	1
B	3
A	2
C	2
B	2

Esto es, ha seleccionado cada fila con los valores x e y diferentes. Ya no aparece duplicado A1, ni C2. Pero lo que queremos obtener, la suma por cada X, sería con la siguiente consulta:

```
select x, sum(y) from T group by x;
```

Cuyo resultado es:

x	y
A	4
B	5
C	4

Podemos entender que cuando hacemos un group by ya implícitamente se selecciona solo un valor diferente de la columna o columnas que hemos indicado en el group by. Y posteriormente se suma o se realiza la operación indicada de forma separada para cada línea diferente. Esto es, si los valores diferentes son A, B y C, tendremos una fila de cada. Para A sumará todas las columnas "y" y el resultado se indicará a su lado. Posteriormente hará lo mismo con el resto de valores: B y C.

6.3. Sintax error ... near ' '

Tenemos la siguiente consulta de la base de datos bd_empresa que da error:

```
select nombre_empleado from t_empleados order by substring(nombre_empleado,locate(", ",nombre_empleado);
```

El error que obtenemos en este caso es siempre el siguiente:

```
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '' at line 1
```

Si nos fijamos, el lugar donde encuentra el error es '', de lo que se deduce que es justo al final. Significa que decimos que la consulta acaba donde ponemos el ; pero por sintaxis le falta algo.

En este caso es un paréntesis de cierre.