**Consultas con varias tablas**

Uno de los temas que más cuesta a los que empiezan a aprender SQL son las consultas en las que se recogen diferentes tipos de datos de una o múltiples tablas. Esta práctica es una introducción a cómo definir consultas de este tipo en PostgreSQL.

Vamos a recordar brevemente algunos conceptos de álgebra relacional que nos pueden ayudar a entender mejor el tema que estamos tratando y que se han estudiado en las clases de teoría.

**Algebra relacional**

El álgebra relacional es un tipo de álgebra con una serie de operadores que trabajan sobre una ó varias relaciones para obtener una relación resultado. Es la base indispensable para poder escribir buenas consultas en SQL.

Las operaciones más importantes disponibles en álgebra relacional son:

* **Operaciones de conjunto** aplicadas a relaciones: unión(∪), intersección(∩) y diferencia(-)
* **Operaciones de eliminación** de una parte de las relaciones: selección (σ) y proyección (Π)

#### Operaciones de combinación de las tuplas de 2 relaciones: producto cartesiano(x), reunión theta (xθ) y reunión natural (⋈)

* **Operación de renombramiento (alias)** que cambia el nombre de los atributos o de la relación: renombre (ρ)

A continuación vamos a dar una breve introducción sobre estas operaciones:

**Unión [R∪S]**

La unión de R y S es el conjunto de elementos que existen en R, ó en S, ó en las dos. Un elemento que existe tanto en R como en S aparece solamente una vez en la unión. En el lenguaje SQL este tipo de operación se puede realizar con la cláusula *UNION*

**Intersección [R∩S]**

La intersección de R y S es el conjunto de elementos que existen en R y en S. En el lenguaje SQL este tipo de operación se puede realizar con la cláusula *INTERSECT*

**Diferencia [R-S]**

La diferencia de R y S es el conjunto de elementos que existen en R pero no en S. R-S es diferente a S-R, S-R seria el conjunto de elementos que existen en S pero no en R. En el lenguaje SQL este tipo de operación se puede realizar con la cláusula *EXCEPT*

**Selección [σc(R)]**

Esta operación aplicada a una relación R, produce una nueva relación con un subconjunto de tuplas de R. Este subconjunto de tuplas satisface y cumple cierta condición(c)

**Proyección [Π a1,a2,...,an(R)]**

Esta operación aplicada a una relación R, produce una nueva relación con solamente los atributos (columnas) especificados por a1,a2,...,an

**Producto cartesiano [RxS]**

El producto cartesiano de dos relaciones R y S es la relación que se obtiene de la combinación de todas las tuplas de R con todas las tuplas de S. Las tuplas de la relación que se obtiene están formadas por todos los atributos de R seguidos de todos los atributos de S. En el lenguaje SQL este tipo de operación se puede realizar con la cláusula *CROSS JOIN* ó separando las relaciones usadas en el producto con comas, en el FROM de la sentencia SQL.

**Combinaciones (JOIN)**

Por medio del operador combinación (JOIN) podemos combinar dos relaciones según una condición para obtener tuplas compuestas por atributos de las dos relaciones combinadas.

En el lenguaje SQL existen diferentes maneras de combinar dos relaciones. A continuación se muestra un resumen de las existentes en PostgreSQL:

* **Combinaciones internas (R INNER JOIN S):** Un INNER JOIN entre dos relaciones R y S, es el resultado que se obtiene después de aplicar al producto cartesiano de las dos relaciones R y S, una condición para acotar dicho producto. Existen un par de casos especiales:
  + **De equivalencia (Equi-join):** Es un caso particular de INNER JOIN en el que la condición que acota el resultado es una comparación de igualdad.
  + **R NATURAL JOIN S:** Es un caso especial de equi-join en el que en el caso de existir columnas con el mismo nombre en las relaciones que se combinan, solo se incluirá una de ellas en el resultado de la combinación.
* **Combinaciones externas (OUTER JOINS):** Un OUTER JOIN entre dos relaciones R y S contiene todas las tuplas que un INNER JOIN devolvería, más una serie de tuplas que no tienen atributos en común en las dos relaciones. Los diferentes tipos son:
  + **R LEFT OUTER JOIN S:** Un LEFT OUTER JOIN entre dos relaciones R y S, retorna todas las tuplas de la combinación que tengan un atributo común, más todas las tuplas de la relación de la izquierda (R) que no tengan un equivalente en la relación de la derecha (S).
  + **R RIGHT OUTER JOIN S:** Un RIGHT OUTER JOIN entre dos relaciones R y S, retorna todas las tuplas de la combinación que tengan un atributo común, más todas las tuplas de la relación de la derecha (S) que no tengan un equivalente en la relación de la izquierda (R).
  + **R FULL OUTER JOIN S:** Un FULL OUTER JOIN entre dos relaciones R y S, retorna todas las tuplas de la combinación que tengan un atributo común, más todas las tuplas de la relación de la izquierda (R) que no tenga un equivalente en la relación de la derecha (S) y todas las tuplas de la relación de la derecha (S) que no tenga un equivalente en la relación de la izquierda (R).

**Renombre [ρS (R)], [ρa/b (R)]**

ρS (R),esta operación ρ aplicada a una relación R**,** produce una nueva relación idéntica a R en donde el nombre de R se ha cambiado por S.

ρa/b (R),esta operación ρ aplicada a una relación R, produce una nueva relación idéntica a R en donde el atricalubuto 'b' ha sido renombrado a 'a'.

El uso combinado de todas estas operaciones aplicado a nuestras relaciones dará lugar a consultas más o menos complejas.

Utilizando los operadores definidos en algebra relacional, podemos definir de manera gráfica (árboles) o lineal la representación algebraica de nuestra consulta. En consultas muy complicadas es lo que se debería de hacer antes de empezar a escribir el código SQL.

**Ejemplos prácticos**

Lo primero que vamos a hacer es definir una BD ‘alge’ con 2 tablas relación\_r y relación\_s que utilizaremos para ilustrar los operadores:

postgres=# SELECT \* FROM relacion\_r;

a | b | c

---+---+---

1 | 2 | 3

4 | 5 | 6

(2 rows)

postgres=# SELECT \* FROM relacion\_s;

c | d | e

---+---+---

4 | 5 | 6

7 | 8 | 9

(2 rows)

En nuestro primer ejemplo realizamos una unión de las 2 relaciones, el resultado obtenido sería:

a | b | c

---+---+---

4 | 5 | 6

1 | 2 | 3

7 | 8 | 9

(3 rows)

A continuación realizamos una intersección entre las 2 relaciones, el resultado sería:

a | b | c

---+---+---

4 | 5 | 6

(1 row)

La diferencia entre estas 2 relaciones daría el siguiente resultado (como se puede comprobar, y por definición, no es lo mismo la diferencia entre relación\_r y relación\_s, que entre relación\_s y relación\_r):

/\* Resta de relacion\_r menos relacion\_s (r-s) \*/

a | b | c

---+---+---

1 | 2 | 3

(1 row)

/\* Resta de relacion\_s menos relacion\_r (s-r) \*/

c | d | e

---+---+---

7 | 8 | 9

(1 row)

Insertaremos una nueva fila (3,4,5) en la tabla relación\_s para ver unos ejemplos de cómo combinar estas dos relaciones mediante JOINs.

postgres=# SELECT \* FROM relacion\_r;

a | b | c

---+---+---

1 | 2 | 3

4 | 5 | 6

(2 rows)

postgres=# SELECT \* FROM relacion\_s;

c | d | e

---+---+---

4 | 5 | 6

7 | 8 | 9

3 | 4 | 5

(3 rows)

La manera más simple de combinar estas 2 relaciones es realizar el producto cartesiano de ambas. Esto se puede realizar de dos maneras, o bien definiendo las dos relaciones separadas por comas después del FROM. El resultado:

a | b | c | c | d | e

---+---+---+---+---+---

1 | 2 | 3 | 4 | 5 | 6

1 | 2 | 3 | 7 | 8 | 9

1 | 2 | 3 | 3 | 4 | 5

4 | 5 | 6 | 4 | 5 | 6

4 | 5 | 6 | 7 | 8 | 9

4 | 5 | 6 | 3 | 4 | 5

(6 rows)

O utilizando la cláusula CROSS JOIN entre las dos relaciones. El resultado:

a | b | c | c | d | e

---+---+---+---+---+---

1 | 2 | 3 | 4 | 5 | 6

1 | 2 | 3 | 7 | 8 | 9

1 | 2 | 3 | 3 | 4 | 5

4 | 5 | 6 | 4 | 5 | 6

4 | 5 | 6 | 7 | 8 | 9

4 | 5 | 6 | 3 | 4 | 5

(6 rows)

En realidad un CROSS JOIN es el equivalente a un INNER JOIN ON (true). Esto porque el INNER JOIN es por definición un producto cartesiano al que se le aplica una condición para acotar el resultado. En este caso particular la condición siempre se cumple para todas las tuplas al utilizar el valor TRUE, con lo que obtendremos todas las tuplas del producto cartesiano. El resultado:

a | b | c | c | d | e

---+---+---+---+---+---

1 | 2 | 3 | 4 | 5 | 6

1 | 2 | 3 | 7 | 8 | 9

1 | 2 | 3 | 3 | 4 | 5

4 | 5 | 6 | 4 | 5 | 6

4 | 5 | 6 | 7 | 8 | 9

4 | 5 | 6 | 3 | 4 | 5

(6 rows)

A continuación se puede ver cómo definir un INNER JOIN con la condición definida dentro de ON (...). Este ejemplo es un Equi-join al utilizarse una

comparación de igualdad en la condición. El resultado:

a | b | c | c | d | e

---+---+---+---+---+---

1 | 2 | **3** | **3** | 4 | 5

(1 rows)

El mismo resultado obtenido con la cláusula INNER JOIN se podría haber conseguido obteniendo el producto cartesiano de las dos relaciones y aplicando una condición con WHERE (definición de INNER JOIN). El resultado:

/\* Producto cartesiano con las relaciones separadas por comas y con WHERE \*/

a | b | c | c | d | e

---+---+---+---+---+---

1 | 2 | **3** | **3** | 4 | 5

(1 rows)

/\* Producto cartesiano con CROSS JOIN y con WHERE \*/

a | b | c | c | d | e

---+---+---+---+---+---

1 | 2 | **3** | **3** | 4 | 5

(1 row)

Un NATURAL JOIN retorna el mismo resultado que un equi-join, pero sin repetir las columnas comunes. Los resultados:

/\* con NATURAL JOIN de relacion\_r y relacion\_r, y en el SELECT una ‘\*’ \*/

a | b | c | d | e

---+---+---+---+---

1 | 2 | 3 | 4 | 5

/\* con NATURAL JOIN de relacion\_r y relacion\_r, y en el SELECT puestos los atributos a,b,c,d,e \*/

a | b | c | d | e

---+---+---+---+---

1 | 2 | 3 | 4 | 5

(1 row)

Como se puede ver, existen diferentes maneras de obtener un mismo resultado utilizando diferentes tipos de consultas. Teniendo los conceptos claros y con práctica, os aseguro que todos estos tipos de consultas os saldrán de forma natural después de un tiempo.

A continuación vamos a ver las combinaciones de tipo OUTER JOIN. En PostgreSQL el uso de la palabra OUTER es opcional.

La primera es un LEFT OUTER JOIN de relación\_r con relación\_s. El resultado es:

a | b | c | c | d | e

---+---+---+---+---+---

1 | 2 | **3** | **3** | 4 | 5

4 | 5 | 6 | | |

(2 rows)

Seguido de un RIGHT OUTER JOIN de relación\_r con relación\_s. El resultado es:

a | b | c | c | d | e

---+---+---+---+---+---

1 | 2 | **3** | **3** | 4 | 5

| | | 4 | 5 | 6

| | | 7 | 8 | 9

(3 rows)

Y para terminar un FULL OUTER JOIN de relación\_r con relación\_s. El resultado es:

a | b | c | c | d | e

---+---+---+---+---+---

1 | 2 | **3** | **3** | 4 | 5

| | | 4 | 5 | 6

4 | 5 | 6 | | |

| | | 7 | 8 | 9

(4 rows)

**Un ejemplo: BD ‘PCs’**

Los ejemplos que hemos visto hasta ahora, han mostrado la sintaxis básica de las operaciones que se pueden utilizar para obtener resultados con datos de múltiples relaciones. En la vida real nos encontraremos con casos muchos más complicados en los que tendremos que combinar todas estas operaciones junto con el resto de operadores y cláusulas SQL disponibles.

En casos complicados es importante pensar antes de empezar a escribir nuestra consulta SQL. A continuación vamos a ver un ejemplo un poco más complicado para ver cómo podemos desglosar y resolver la consulta que necesitamos para obtener el resultado deseado.

Crearemos desde psql la BD con nombre PCs (con CREATE DATABASE) y nos conectamos con la BD recién creada:

template1=# CREATE DATABASE pcs;

template1=# \c pcs

pcs=#

Utilizaremos unas tablas creadas para este ejemplo (con CREATE TABLE). Tenemos una tabla principal llamada 'PC', con diferentes columnas conteniendo cadenas de identificación de los diferentes componentes usados para el ensamblado de diferentes modelos de PCs. Las columnas vacías de la tabla 'PC' significan componentes no presentes en dichos modelos. El resto de tablas contienen información adicional sobre los diferentes componentes usados para construir un PC.

postgres=# SELECT \* FROM pc;

pcid | memoria | cpu | disco | tgrafica | precio

------+---------+---------+-----------+-----------+--------

1 | mem0001 | cpu0001 | disco0001 | ati001 | 1000

2 | mem0001 | cpu0001 | disco0002 | ati001 | 1100

3 | mem0002 | cpu0002 | disco0003 | nvidia001 | 1400

4 | mem0004 | cpu0003 | disco0004 | nvidia001 | 1600

5 | | cpu0001 | disco0001 | ati001 | 900

6 | | | | ati001 | 400

(6 rows)

postgres=# SELECT \* FROM cpu ;

cpu\_id | cpu\_fabricante | cpu\_tipo

---------+----------------+------------

cpu0001 | intel | Core2 duo

cpu0002 | intel | Core2 Quad

cpu0003 | amd | Athlon X2

(3 rows)

postgres=# SELECT \* FROM memoria ;

mem\_id | mem\_capacidad | mem\_tipo

---------+---------------+------------

mem0001 | 1024 | DDR SDRAM

mem0002 | 1024 | DDR2 SDRAM

mem0003 | 1024 | DDR3 SDRAM

mem0004 | 2048 | DDR3 SDRAM

(4 rows)

postgres=# SELECT \* FROM disco ;

disco\_id | disco\_fabricante | disco\_capacidad

-----------+------------------+-----------------

disco0001 | seagate | 350

disco0002 | seagate | 500

disco0003 | seagate | 1024

disco0004 | samsung | 500

(4 rows)

postgres=# SELECT \* FROM tgrafica ;

tgraf\_id | tgraf\_fabricante

-----------+------------------

ati001 | ati

nvidia001 | nvidia

Utilizando estas tablas vamos a realizar varias consultas:

**Consulta 1**

*Obtener una relación de solo los modelos de PC* ***'completos'*** *a la venta. Queremos toda la información disponible sobre los componentes que lo forman (pcid, mem\_tipo, mem\_capacidad, cpu\_fabricante, cpu\_tipo, disco\_fabricante, disco\_capacidad, tgraf\_fabricante, precio). Ordenar el resultado de mayor a menor precio (con la cláusula ORDER BY).*

Sabemos que al tener que coger datos de diferentes tablas, necesitaremos algún tipo de cláusula JOIN. En esta consulta nos piden *solamente*, PCs completos, con todos sus componentes. Por ello descartamos todas las combinaciones de tipo OUTER JOIN y utilizamos INNER JOIN para obtener solamente las tuplas con atributos en todas las relaciones combinadas.

Cada PC tiene 4 componentes y la información de cada componente se encuentra en una tabla separada. Con estos datos sabemos que tendremos que realizar 4 INNER JOIN entre todas las tablas involucradas en la consulta.

Vamos a empezar a escribir la consulta SQL. Primero realizamos los INNER JOIN y declaramos las condiciones que acotarán el resultado. Tendremos que emparejar los atributos de componentes presentes en la tabla 'PC' con los correspondientes atributos en el resto de tablas de componentes. Utilizaremos los alias: a, b, c, d y e para las tablas: pc, memoria, cpu, disco y tgrafica, respectivamente. Inicialmente mostrar todos los atributos de todas las tablas (SELECT \* .... ):

SELECT \*

FROM pc AS a

INNER JOIN memoria AS b ON (a.memoria = b.mem\_id)

INNER JOIN cpu AS c ON (a.cpu = c.cpu\_id)

INNER JOIN disco AS d ON (a.disco = d.disco\_id)

INNER JOIN tgrafica AS e ON (a.tgrafica = e.tgraf\_id);

Una vez que hemos combinado todas las tablas, vamos a mostrar los atributos que queremos presentar en nuestro resultado (pcid, mem\_tipo, mem\_capacidad, cpu\_fabricante, cpu\_tipo, disco\_fabricante, disco\_capacidad, tgraf\_fabricante, precio). Utilizaremos los prefijos de tablas definidos en la consulta anterior (a,b,c,d,e), para acceder a los atributos de cada tabla:

**SELECT**

a.pcid,

b.mem\_tipo,

b.mem\_capacidad AS mem\_MB,

c.cpu\_fabricante AS cpu\_fab,

c.cpu\_tipo,

d.disco\_fabricante AS disco\_fab,

d.disco\_capacidad AS disco\_GB,

e.tgraf\_fabricante AS tgraf\_fab,

a.precio

**FROM** pc AS a

INNER JOIN memoria AS b ON (a.memoria = b.mem\_id)

INNER JOIN cpu AS c ON (a.cpu = c.cpu\_id)

INNER JOIN disco AS d ON (a.disco = d.disco\_id)

INNER JOIN tgrafica AS e ON (a.tgrafica = e.tgraf\_id);

Y para terminar ordenamos el resultado por el precio (con ORDER BY):

SELECT

a.pcid,

b.mem\_tipo,

b.mem\_capacidad AS mem\_MB,

c.cpu\_fabricante AS cpu\_fab,

c.cpu\_tipo,

d.disco\_fabricante AS disco\_fab,

d.disco\_capacidad AS disco\_GB,

e.tgraf\_fabricante AS tgraf\_fab,

a.precio

FROM pc AS a

INNER JOIN memoria AS b ON (a.memoria = b.mem\_id)

INNER JOIN cpu AS c ON (a.cpu = c.cpu\_id)

INNER JOIN disco AS d ON (a.disco = d.disco\_id)

INNER JOIN tgrafica AS e ON (a.tgrafica = e.tgraf\_id)

ORDER BY precio DESC;

El resultado de nuestra consulta presentará las características de solo los PC completos:

pcid | mem\_tipo | mem\_mb | cpu\_fab | cpu\_tipo | disco\_fab | disco\_gb | tgraf\_fab | precio

------+------------+--------+---------+------------+-----------+----------+-----------+--------

4 | DDR3 SDRAM | 2048 | amd | Athlon X2 | samsung | 500 | nvidia | 1600

3 | DDR2 SDRAM | 1024 | intel | Core2 Quad | seagate | 1024 | nvidia | 1400

2 | DDR SDRAM | 1024 | intel | Core2 duo | seagate | 500 | ati | 1100

1 | DDR SDRAM | 1024 | intel | Core2 duo | seagate | 350 | ati | 1000

(4 rows)

**Consulta 2**

*Obtener una relación de todos los modelos de PC a la venta. Queremos toda la información disponible sobre los componentes que lo forman (pcid, mem\_tipo, mem\_capacidad, cpu\_fabricante, cpu\_tipo, disco\_fabricante, disco\_capacidad, tgraf\_fabricante, precio). Ordenar el resultado de mayor a menor precio.*

En esta consulta nos piden lo mismo que la consulta 1 pero de *todos* los modelos de PC, los completos y los que se venden sin algún componente. La tabla PC es la tabla principal de nuestra combinación, y la tabla a la que le faltan valores en ciertos atributos en algunas tuplas. Esta tabla es la primera que se define cuando definimos las cláusulas JOIN y por definición es la que se encuentra más a la izquierda. Por ello utilizaremos el tipo LEFT OUTER JOIN para conseguir el resultado de la consulta 1, más todos los PC a los que le falta algún componente.

Lo único que tenemos que hacer es cambiar INNER JOIN por LEFT OUTER JOIN. La consulta quedaría así:

**SELECT**

a.pcid,

b.mem\_tipo,

b.mem\_capacidad AS mem\_MB,

c.cpu\_fabricante AS cpu\_fab,

c.cpu\_tipo,

d.disco\_fabricante AS disco\_fab,

d.disco\_capacidad AS disco\_GB,

e.tgraf\_fabricante AS tgraf\_fab,

a.precio

**FROM** pc AS a

LEFT OUTER JOIN memoria AS b ON (a.memoria = b.mem\_id)

LEFT OUTER JOIN cpu AS c ON (a.cpu = c.cpu\_id)

LEFT OUTER JOIN disco AS d ON (a.disco = d.disco\_id)

LEFT OUTER JOIN tgrafica AS e ON (a.tgrafica = e.tgraf\_id)

**ORDER BY** precio DESC;

Y el resultado de nuestra consulta presentará las características de todos los PC:

pcid | mem\_tipo | mem\_mb | cpu\_fab | cpu\_tipo | disco\_fab | disco\_gb | tgraf\_fab | precio

------+------------+--------+---------+------------+-----------+----------+-----------+--------

4 | DDR3 SDRAM | 2048 | amd | Athlon X2 | samsung | 500 | nvidia | 1600

3 | DDR2 SDRAM | 1024 | intel | Core2 Quad | seagate | 1024 | nvidia | 1400

2 | DDR SDRAM | 1024 | intel | Core2 duo | seagate | 500 | ati | 1100

1 | DDR SDRAM | 1024 | intel | Core2 duo | seagate | 350 | ati | 1000

5 | | | intel | Core2 duo | seagate | 350 | ati | 900

6 | | | | | | | ati | 400

(6 rows)

**Consulta 3**

*Obtener una relación de solo los modelos de 'PC* ***NO completos****' a la venta. Queremos toda* ***la información disponible*** *sobre los componentes que lo forman*

Si os fijáis, esta consulta la podríamos obtener si al resultado que muestra *todos* los PCs, le 'restamos' el resultado con *sólo* los PCs completos. Esto lo podríamos realizar combinando la consulta 1 con la consulta 2 mediante el operador EXCEPT (*consulta 2* EXCEPT *consulta 1*):

(

**SELECT \* FROM**

**(**

(

**SELECT**

a.pcid,

b.mem\_tipo,

b.mem\_capacidad AS mem\_MB,

c.cpu\_fabricante AS cpu\_fab,

c.cpu\_tipo,

d.disco\_fabricante AS disco\_fab,

d.disco\_capacidad AS disco\_GB,

e.tgraf\_fabricante AS tgraf\_fab,

a.precio

**FROM** pc AS a

LEFT OUTER JOIN memoria AS b ON (a.memoria = b.mem\_id)

LEFT OUTER JOIN cpu AS c ON (a.cpu = c.cpu\_id)

LEFT OUTER JOIN disco AS d ON (a.disco = d.disco\_id)

LEFT OUTER JOIN tgrafica AS e ON (a.tgrafica = e.tgraf\_id)

)

**EXCEPT**

(

SELECT

a.pcid,

b.mem\_tipo,

b.mem\_capacidad AS mem\_MB,

c.cpu\_fabricante AS cpu\_fab,

c.cpu\_tipo,

d.disco\_fabricante AS disco\_fab,

d.disco\_capacidad AS disco\_GB,

e.tgraf\_fabricante AS tgraf\_fab,

a.precio

FROM pc AS a

INNER JOIN memoria AS b ON (a.memoria = b.mem\_id)

INNER JOIN cpu AS c ON (a.cpu = c.cpu\_id)

INNER JOIN disco AS d ON (a.disco = d.disco\_id)

INNER JOIN tgrafica AS e ON (a.tgrafica = e.tgraf\_id)

)

**) AS con**

**ORDER BY precio DESC;**

OTRA forma:

**SELECT \* FROM**

**(**

(**SELECT** a.pcid **FROM** pc AS a)

**EXCEPT**

(

**SELECT** a.pcid

**FROM** pc AS a

**INNER** **JOIN** memoria AS b **ON** (a.memoria = b.mem\_id)

**INNER** **JOIN** cpu AS c **ON** (a.cpu = c.cpu\_id)

**INNER** **JOIN** disco AS d **ON** (a.disco = d.disco\_id)

**INNER** **JOIN** tgrafica AS e **ON** (a.tgrafica = e.tgraf\_id)

)

**) AS** con **NATURAL JOIN** pca

**LEFT JOIN** memoria b **ON** (a.memoria = b.mem\_id)

**LEFT JOIN** cpuc **ON** (a.cpu = c.cpu\_id)

**LEFT JOIN** discod **ON** (a.disco = d.disco\_id)

**LEFT JOIN** tgraficae **ON** (a.tgrafica = e.tgraf\_id)

El resultado sería el esperado:

pcid | mem\_tipo | mem\_mb | cpu\_fab | cpu\_tipo | disco\_fab | disco\_gb | tgraf\_fab | precio

------+----------+--------+---------+-----------+-----------+----------+-----------+--------

5 | | | intel | Core2 duo | seagate | 350 | ati | 900

6 | | | | | | | ati | 400

(2 rows)

**Consulta 4**

*Obtener una relación de todos los PC que tengan CPUs de AMD y discos Samsung. Queremos toda la información disponible sobre los componentes que lo forman (pcid, mem\_tipo, mem\_capacidad, cpu\_fabricante, cpu\_tipo, disco\_fabricante, disco\_capacidad, tgraf\_fabricante, precio).*

El trabajo para definir esta consulta está casi hecho. Lo único que tenemos que hacer es aplicar mediante la sentencia WHERE, las dos condiciones que nos piden, a la consulta 2:

**SELECT**

a.pcid,

b.mem\_tipo,

b.mem\_capacidad AS mem\_MB,

c.cpu\_fabricante AS cpu\_fab,

c.cpu\_tipo,

d.disco\_fabricante AS disco\_fab,

d.disco\_capacidad AS disco\_GB,

e.tgraf\_fabricante AS tgraf\_fab,

a.precio

**FROM** pc AS a

LEFT OUTER JOIN memoria AS b ON (a.memoria = b.mem\_id)

LEFT OUTER JOIN cpu AS c ON (a.cpu = c.cpu\_id)

LEFT OUTER JOIN disco AS d ON (a.disco = d.disco\_id)

LEFT OUTER JOIN tgrafica AS e ON (a.tgrafica = e.tgraf\_id)

**WHERE**

LOWER(c.cpu\_fabricante) = 'amd' AND LOWER(d.disco\_fabricante) = 'samsung'

**ORDER BY** precio DESC;

Y el resultado quedaría asi:

pcid | mem\_tipo | mem\_mb | cpu\_fab | cpu\_tipo | disco\_fab | disco\_gb | tgraf\_fab | precio

------+------------+--------+---------+-----------+-----------+----------+-----------+--------

4 | DDR3 SDRAM | 2048 | amd | Athlon X2 | samsung | 500 | nvidia | 1600

(1 row)

**Consulta 5**

*Obtener una relación de PCs (pcid, precio) que tienen CPUs de Intel o de AMD. Ordenar de mayor a menor precio.*

Como queremos obtener solamente los PC que tienen algún tipo de CPU, utilizaremos un INNER JOIN para descartar los PCs sin CPU definida y mostraremos todos los atributos inicialmente.

SELECT \*

FROM pc

INNER JOIN cpu ON pc.cpu = cpu.cpu\_id;

A continuación ponemos la condición que la CPU sea de Intel o de AMD y mostramos sólo pcid y precio:

SELECT pc.pcid, pc.precio

FROM pc

INNER JOIN cpu ON pc.cpu = cpu.cpu\_id

WHERE cpu.cpu\_fabricante = 'amd' OR cpu.cpu\_fabricante = 'intel';

Y ordenamos el resultado por el precio descendentemente:

SELECT pc.pcid, pc.precio

FROM pc

INNER JOIN cpu ON pc.cpu = cpu.cpu\_id

WHERE cpu.cpu\_fabricante = 'amd' OR cpu.cpu\_fabricante = 'intel'

ORDER BY pc.precio DESC;

El resultado obtenido sería:

pcid | precio

------+---------

4 | 1600.00

3 | 1400.00

2 | 1100.00

1 | 1000.00

5 | 900.00

(5 rows)

El objetivo de esta práctica ha sido, cómo funciona el combinar diferentes tablas para obtener un resultado. En la vida real os encontraréis con ejemplos bastantes complicados, lo importante es pensar la estrategia a seguir antes de empezar a escribir la consulta SQL. También es importante hacerlo paso a paso, aplicando las restricciones necesarias hasta conseguir lo que se desea. Solamente la práctica ayudará a entender y escribir con soltura consultas complejas.